2000

# Congestion Control Mechanisms for Internet Multicast Transport Protocols.

Elias George Khalaf
*Louisiana State University and Agricultural & Mechanical College*

# CONGESTION CONTROL MECHANISMS FOR INTERNET MULTICAST TRANSPORT PROTOCOLS

A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

in

The Department of Computer Science

by
Elias G. Khalaf
B.S., University of Southwestern Louisiana, 1989
M.S. in Sy. Sc., Louisiana State University, 1992
May 2000

# Acknowledgments

I would like to thank my advisor Dr. S. S. Iyengar for his guidance in this research effort, and for the freedom and continued support he provided all these years. I also would like to thank all my family and friends who supported me throughout my extended stay at LSU.

# Table of Contents

# List of Tables

# List of Figures

# Glossary of Terms

**ACK** Positive Acknowledgment.

**ALF** Application Level Framing

**DVMRP** Distance Vector Multicast Routing Protocol.

**FIFO** First-In-First-Out. A queue data structure where elements are added on one end of the queue and are removed from the other in the order they were added.

**HACK** Hierarchical Acknowledgment.

**IGMP** Internet Group Management Protocol.

**IP** Internet Protocol.

**LGC** Local Group Concept

**MBone** Multicast Backbone Network.

**NACK** Negative Acknowledgment.

**RAMP** Reliable Adaptive Multicast Protocol.

**RFC** Request For Comment. An Internet document used to gather data on standards and other information relevant to the Internet.

**RINA** Receiver-Initiated with NACK Avoidance.

**RMTP** Reliable Multicast Transport Protocol.

**RTP** Reliable Transport Protocol.

**SRM** Scalable Reliable Multicast. A reliable multicast transport protocol.

**TCP** Transmission Control Protocol.

**XTP** The Xpress Transfer Protocol.

# Abstract

Internet Multicasting has emerged in the last few years as the inevitable method for efficiently delivering replicated data to multiple recipients in large scale internets. More specifically, Reliable Multicast Transport Protocols have been the subject of intensive research in recent years. The goal of such research was mainly to develop reliable and scalable protocols to efficiently and reliably deliver data to receivers, and at the same time reduce congestion on the network. In this research, we propose a new generic family of protocols, called the $K$ family of protocols, that focus on congestion control by reducing the processing requirements on receivers to $O(1)$, while being able to tune the processing requirements on the sender to the sender's processing capability. The concept of Local Recovery is then applied to the $K$ protocol family, achieving even further improvement in processing requirements, especially at the sender. In all cases, processing requirements at the sender and at the receivers are analytically studied. We also introduce a new concept that attempts to detect packet loss and repair it before control packets cause implosions at the sender or at the receivers. Finally, some numerical results are presented to show the relative reduction in processing requirements in comparison with other prominent generic classes of protocols.

# Chapter 1

# Introduction

Internet Multicasting has emerged in the last few years as the premier foreseeable method for efficiently delivering replicated data to multiple recipients in large scale internets. Multicasting has been fueled by the following factors:

- the need to efficiently and reliably deliver replicated data on the Internet to multiple recipients. Examples include audio and video, replicated databases, real-time and multimedia applications, streaming data applications, etc.
- replicated data streams will eventually become a limiting factor in terms of bandwidth and reliability on the Internet. New methods had to be created to resolve congestion and replication problems.

In the late 80's and early 90's multicasting took off as the solution to the problems described above. The pioneering work of [6] paved the way for multicasting as a standard that eventually found its way into the Internet protocol and has become an integral part of it.

## 1.1 Reliable Transport Protocols

The area of reliable multicast transport protocols has been a very active research area of the past few years. Many researchers have been attempting to create one generic protocol that will be suitable for all applications. It has been realized, however, that there is no one protocol that serves this purpose. Applications must have greater control over the segmentation and framing of data units. This area of research is still being worked on without any major success. On the other hand,

there has been a flood of new multicast protocols, each trying to serve a different multicasting need or trying to improve on an existing protocol. Several problems still exist in this area, one of which is that of *congestion control.* This is the main focus of this research.

## 1.2 Scope of the Dissertation

In this research we approach the problem of congestion control in Internet Multicast Transport Protocols. The problem is treated from a different perspective than the classical network bandwidth based approach. Instead we focus on processing requirements at the sender and at the receivers, with the conjecture that network bandwidth will keep outgrowing processing speeds for the next few years. Therefore, in this research, we devise new schemes for congestion control in multicast enabled internets that minimize processing requirements on receivers (which constitute the bulk of the network) as well as reduce or tune those requirements on senders. We devise a new approach to error recovery in multicast transport protocols, and then we combine it with the well known approach of Local Recovery to achieve even greater reduction in processing requirements.

We also introduce a new concept that attempts to predict great packet loss in the network and tries to remedy it before its cascading effects congest the network.

Finally, we analyze the performance of our proposed new schemes in comparison with other well known schemes.

## 1.3 Outline

Chapter 2 introduces the reader to the area of Internet Multicasting, with special concentration on Internet Multicast Transport Protocols. In that chapter we survey some of the most prominent protocols, and then we present a general classification of multicast protocols along with their expected analytical performance complexities.

In chapter 3 we introduce our new concept of Global Pairing, describe its mechanics, and propose a new protocol, the $K$ protocol that utilizes Global Pairing. We then analyze the processing requirements on the sender and on the receiver under $K$. In chapter 4 we apply the concept of Local Recovery to Global Pairing and call the combined concept Local Pairing. A new protocol, the $KL$, is then described. We then perform an analysis similar to the one in the previous chapter. In chapter 5 we generalize the concept of Global Pairing and we call it Grouping of receivers. Again, we propose the $K'$ protocol and then apply local recovery schemes to produce the $KL'$ protocol. We even go one step further by applying grouping to repair servers themselves, further improving processing costs at the sender. In chapter 6 we describe briefly the new concept of Preemptive Multicasting, which is aimed at detecting great losses in the network and remulticasting the lost segments as a measure to prevent control message and repair implosions. Finally, in chapter 7 we plot the complexities of the $K'$ protocol with respect to that of a similar-class protocol. The chapter ends with a discussion on the pros and cons of the different classes of protocols and the choices to make. Chapter 8 summarizes our research and sheds some light on possible future expansion of this work.

# Chapter 2

# Internet Multicasting

## 2.1 Introduction

With the explosive growth of the Internet, there is growing demand for delivering different types of data for different purposes. The classical *point-to-point* communication scheme is used by most of today's Internet applications. However, new applications are arising that require delivery of data from single or multiple sources to multiple recipients. Examples include audio and video delivery, distributed file replication, resource discovery, multimedia conferencing, shared white boards, among others.

*Multicasting* refers to the delivery of data packets from a source (*sender*) to a set of destinations (*receivers*), rather than just a single destination [6]. Data packets can be sent from the sender to each receiver separately, but that would be wasteful of network bandwidth and of the sender's processing power. Instead, using multicasting, network routers perform replication of data when necessary to eventually reach all destinations. The sender in this case only sends out one copy of the data. Multicasting is depicted in Figure 2.1 as opposed to other modes of communication, namely *Broadcast*, *Unicast* and *Replicated Unicast*.

Much research has been devoted recently to devising *reliable* and *scalable* multicast protocols that would work well for most applications, and a number of reliable multicast protocols have been proposed for wide area packet networks. However, it has been shown that one cannot make a single reliable multicast delivery scheme

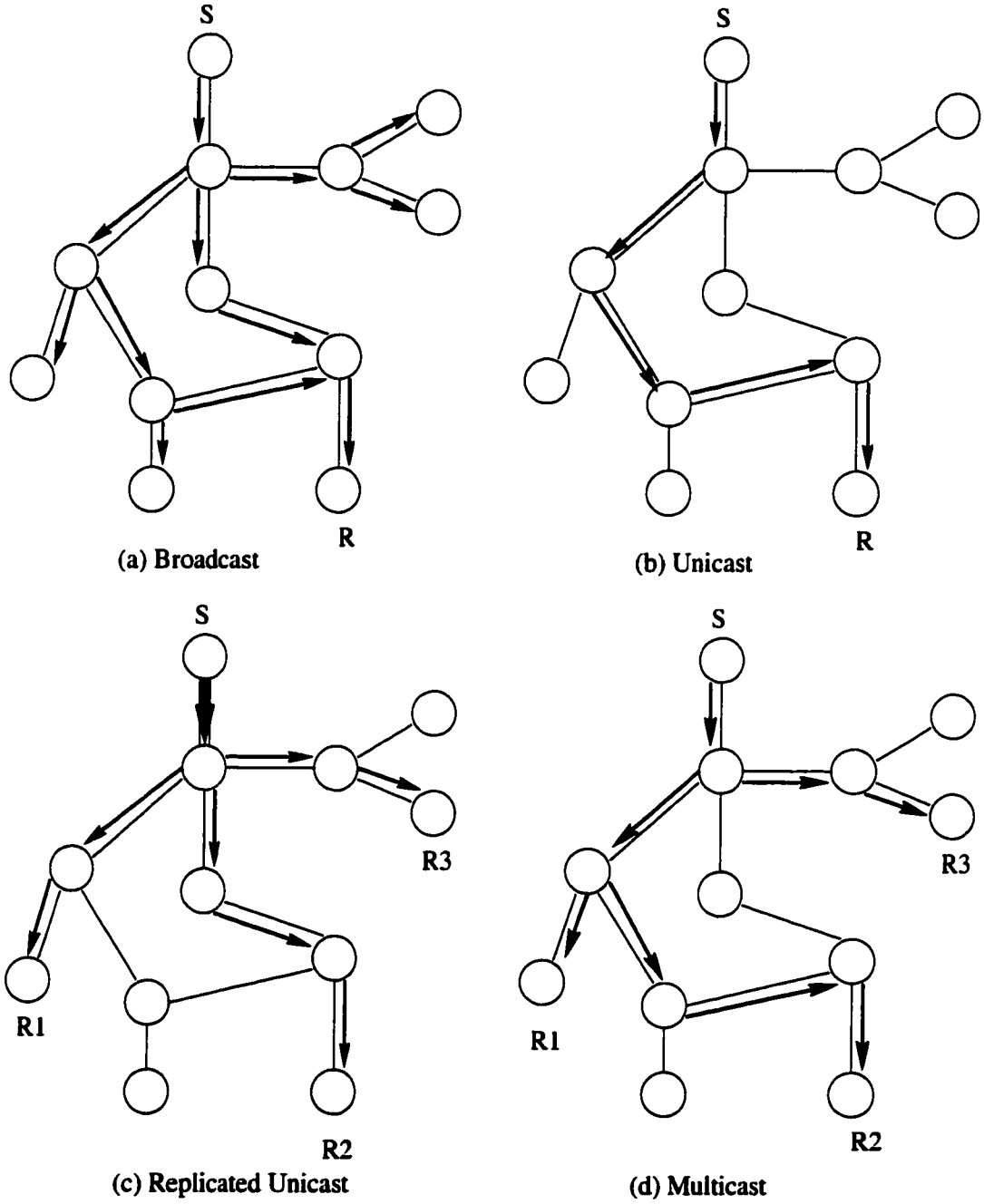



Figure 2.1: Basic Communication Models

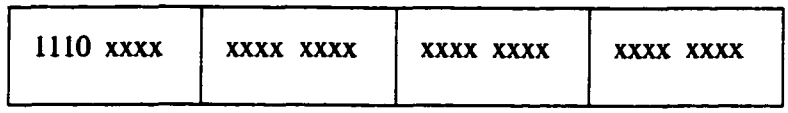



Figure 2.2: Class D Address

that optimally meets the functionality, scalability, and efficiency requirements of all applications [8].

## 2.2 Multicasting

Multicasting refers to the transmission of data packets from a source, called a *sender*, to a number of destinations, called *receivers*. The receivers, collectively, make up a *multicast group*. Data packets are routed from the sender to the receivers by network elements called *routers*. During a multicast session, information is distributed from a sender to the receivers in its multicast group [3]. Although data could be sent independently from the source to each destination, this would be very inefficient since it consumes network bandwidth wastefully and may create congestion if the number of receivers is high.

### 2.2.1 IP Multicast

This refers to the multicasting technique described above, used in the network layer of the Internet. Just like classes A, B and C of the Internet Protocol (IP), there exists a class D to address IP multicast groups. A class D IP address has a "1110" in the four most significant bit positions of the 32-bit address used for Internet hosts as depicted in Figure 2.2. Therefore, class D IP addresses will fall in the range 224.0.0.0 to 239.255.255.255, inclusive [11, 14]. There is a potential for having 268,435,456 unique multicast groups!

The idea behind class D addressing is that the sender need only know the IP address of the multicast group. It then sends an IP packet as if it were destined to a class A, B or C host. An example of a class D address is 225.30.2.1.

### 2.2.2 IP Multicast Routing

IP multicast routing consists of two protocols: the distance vector multicast routing protocol (DVMRP) and the Internet group management protocol (IGMP) [6]. DVMRP, which executes in multicast-capable routers, builds and maintains a multicast tree, rooted at the sender. The routers use the multicast tree to route multicast packets to all receivers. DVMRP uses distance vectors, representing the distance (number of hops or number of intermediate routers) between the sender and a router, plus knowledge of the multicast group membership to construct a tree with the minimum number of hops from the sender to all receivers. This method of multicasting makes efficient use of network resources. IGMP is used between a receiver and its nearest router to indicate the receiver's interest in joining or leaving the multicast group. Changes in multicast group membership trigger updates to the multicast tree.

A router that receives data packets destined for a multicast group identified by the multicast group address forwards copies of the packets along all multicast tree branches emanating from the router. It also forwards the packets to all receivers directly connected to it. Only one copy of a packet is forwarded along each branch of the multicast tree.

### 2.2.3 MBone: The Multicast Backbone

In 1994, several routers in the Internet were made multicast-capable. They were then connected to each other using *IP tunnels* to implement a logical multicast net-

work on top of a physical network that consisted mainly of unicast routers. Called the Multicast Backbone [17], or MBone, this multicast network is in widespread use today. Voice, video, and whiteboard applications are common applications that use the MBone. With the promise of an increasing need for multicasting capabilities, switch vendors such as Cisco and Bay Networks now include the IP Multicast protocols DVMRP and IGMP in their default router configurations.

## 2.3 Reliable Multicast Transport Protocols

At the network layer, multicast packet delivery is a datagram service, as is unicast packet delivery. This means that multicast packet delivery is also connectionless, provides no guarantee of reliability, and does not promise sequential delivery. The only difference is that unicast packets only go to at most one place, whereas multicasts go where the receivers are, and they are not treated specially by the network [20].

### 2.3.1 Reliability

Reliability, at its most basic meaning, means delivery of the same original data, in the same order it was sent, no matter how much time it took. Here we cannot make any assumptions about delivery trees, except that they are usually highly heterogeneous. You can typically find links of different speeds across the Internet, routers with different memory, processor and buffer space, and different levels of congestion. These differences will most likely exist across any multicast group's members, which poses a serious threat to any reliable multicast transport protocol.

### 2.3.2 Requirements of Applications for Reliability

Different multicast applications have disparate requirements for reliability [20]. Each of these requirements has different design implications for a reliable multicast protocol.

- Some applications require delivery of data by all members regardless of order or time it takes.
- Some applications require total ordering and reliable reception.
- Some require many or all of the members to send data.
- Some support small groups, and some thousands of receivers.
- Some need timely delivery of data.

It is generally agreed that a single reliable multicast framework is not likely to meet the needs of all applications over groups of any size. It is very difficult to know how to map between application-specific requirements and the many potential reliable multicast transport mechanisms [20].

### 2.3.3 Desirable Features for Reliable Multicast Protocols

Reliable multicast protocols should not be reduced to sending a packet as many times as needed to get to its destination. That would be an overkill of network bandwidth, and would not take advantage of any spatial locality of reference.

Here are some of the characteristics that researchers would like to see in a reliable multicast protocol:

- Avoiding overburdening the sender with control traffic retransmissions.
- Being able to handle delivery to heterogeneous groups fairly and efficiently.
- Exhibiting good performance.
- Capability to handle large groups.
- Minimizing control traffic.

- Capability to handle mixed-mode (reliable and unreliable) transport to the same group.
- Welcomes late joiners.

As mentioned earlier, it is unlikely that there will be a universal "one-size-fits-all" multicast reliable transport protocol. Instead, several "classes" of multicast reliable protocols will be developed to match each application's desired reliability. Keep in mind that some applications simply do not require reliable delivery and can tolerate small amounts of loss.

## 2.4 Survey of Major Multicast Protocols

The following host of protocols, few of which have been actually incorporated into a distributed system and tested, are mostly experimental and are in various stages of development. We present a general description of each of the major protocols and point out the advantages and disadvantages of each.

Reliable multicast protocols have two classifications: generalized and specialized. Under each of those categories we will list a host of protocols that have been documented in the literature.

### 2.4.1 Generalized Reliable Multicast Protocols

#### Scalable Reliable Multicast (SRM)

Floyd et al described SRM in [8]. SRM is targeted at operation over a future Internet in which multicast is very common. It has been demonstrated (prototyped) in *wb*, a distributed whiteboard application, which has been used on a global scale with sessions ranging from a few to a few hundred participants.

The design goals of SRM were to be able to work with groups that are potentially very large; to not require special support from IP; and to not interfere with the

operation of TCP, the primary unicast reliability mechanism. Finally SRM had the aspiration to operate efficiently and to adapt to any dynamic changes to group membership. The burden of reliability in SRM has been shifted to the receiver and the *application level framing* protocol model.

In 1990, Clark and Tennenhouse [5] proposed a new protocol model called Application Level Framing (ALF) which explicitly includes an application's semantics in the design of that application's protocol. This way, applications have more control over their communication. All data is exchanged as Application Data Units (ADUs), with data expressed in an application-specific name space and with the application's data encoded to suit its reliability needs.

SRM only guarantees that all the data will be sent to the group. It is not expected to deliver data in any particular order (e.g. the order sent). If the need arises, machinery to enforce a particular delivery order can be easily added on top of this reliable delivery service. SRM, however, is not perfect. In [3] it is noted that SRM has two drawbacks that affect its scalability.

The first drawback is that all requests for and retransmissions of missing data are always re-multicasted to *all* receivers. This wastes valuable network bandwidth and causes unnecessary processing of retransmission requests at all receivers.

The second drawback to SRM scalability is that each receiver must keep track of all other receivers in the multicast group. This is necessary to support the SRM retransmission scheme that allows any receiver to retransmit requested data. As the number of receivers increases, it becomes unwieldy for each receiver to keep track of all other receivers. One mechanism to help localize the scope of requests and repairs is the Local Group Concept (LGC), which is discussed in the next section.

### Local Group Concept (LGC)

This concept was proposed in [12], where the multicast group is divided into Local Groups, each represented by a Group Controller that handles retransmissions for members in the Local Group. The Group Controller is not a router or a separate server, but simply one of the members of the multicast group. Hofmann in [12] aims at the dynamic generation of Local Groups and of Group Controllers, but does not explore in detail the algorithms for finding the nearby Local Group, responding to the failure of a local Group Controller, or choosing a new Group Controller [8].

### Reliable Multicast Transport Protocol (RMTP)

RMTP is being developed by researchers [19] at AT&T's Bell Laboratories, a division of Lucent Technologies. The reliability service it provides is ordered and lossless. Like SRM, it also includes among its goals scalability and receiver-based reliability. RMTP accomplishes this by using Designated Routers (DRs) in each region of the multicast group, where the DRs receive incoming status messages from the region, indicating which packets have been successfully received, and performs retransmissions as needed. The unique feature of RMTP is its hierarchical organization, with regional aggregation of status messages. This prevents the sender from being inundated with status messages, and minimizes the number of group-wide repair messages. The problem of dynamically choosing DRs for a given multicast tree is left for continued research.

## 2.4.2 Specialized Reliable Multicast Protocols

### Reliable Adaptive Multicast Protocol (RAMP)

RAMP was originally described in RFC 1458 [2] and was designed for military collaborative applications such as simulated war games. Consequently, it was designed

to provide low latency and reliable delivery to all recipients. It was also developed to operate at very high speeds over the ARPA-sponsored Testbed for Optical Networking (TBONE) project, an optical, circuit-switched network operating initially at 800Mbps. What is different in this environment is that congestion in the network is almost nonexistent and packet loss mainly comes from host buffer overflow in receivers. The sender under this protocol has complete knowledge of all group members. RAMP has the ability to support two unreliable modes of operation. To date, RAMP has been confined to military application [21].

### RMTP+

RMTP+, another reliable multicast transport protocol also developed at AT&T Bell Labs [3], is an extension of RMTP that reliably multicasts continuous data streams, rather than files, from a sender to an unknown number of receivers.

### Multicast File Transfer Protocol (MFTP)

MFTP was first publicly documented in an Internet Draft in February 1997 and a new Internet Draft was submitted to the IETF in April 1998 [22]. It was originally developed by StarBust Communications and refined with the help of engineering personnel from Cisco Systems. Many WAN products are based on MFTP. MFTP is simple and operates with all network infrastructures, but it is narrowly focused on non-real-time reliable delivery. RMTP has some nice hierarchy characteristics that provide scaling and has been adapted to cover virtually all applications.

There are many more reliable multicast protocols. However, the most common features of their designs have already been exhibited above.

## 2.5 Performance Analysis of Multicast Protocols

The first analytical attempt in [25] and [28] to study the performance of reliable multicast protocols paved the way for researchers in this area. In [25], the performance of two fundamental classes of reliable multicast protocols were compared, namely, the sender-initiated and the receiver-initiated classes. Many other protocols were designed that belonged to neither categories. In effect, two more categories were added by [18], namely the ring-based protocols and the tree-based protocols.

Most of the analysis and results in this section are taken from the above mentioned work and from the compilation by [23].

### 2.5.1 Broad Categories of Reliable Multicast Protocols

In the last few years, there has been a flurry of research activities on reliable multicast transport protocols. As we have seen before, there are numerous protocols most of which are still experimental and very few have been implemented and tested. Regardless of the number of such protocols, they can be categorized broadly into four different classes [18]:

1. Sender-initiated
2. Receiver-initiated
3. Ring-based
4. Tree-based

This classification was based on how the different protocols adjust the *memory allocation window (mw)* and the *congestion window (cw)*. The memory allocation window is concerned with releasing of buffer associated with a block of data while congestion window is associated with the rate of transmission or retransmission.

### Sender-initiated Protocols

In this class of protocols, the sender keeps track of the state of each receiver. Receivers send positive acknowledgements (ACKs) directly to the sender and only after receiving ACKs from *all* the receivers can the sender decide whether to advance the memory allocation window (*mw*) or not. These protocols suffer from the well-known ACK-implosion problem meaning that the sender has to process an excessive number of ACKs, which increases the sender's processing load. An example of sender-initiated protocols is XTP [26].

### Receiver-initiated Protocols

In this class of protocols, the sender does not keep track of the receivers, but rather the receivers themselves send negative acknowledgments (NACKs) to the sender when they detect a missing packet. The sender retransmits the missing packets. However, if each receiver sends a NACK to the sender, that may lead to a NACK-implosion at the sender. In order to avoid the NACK-implosion, when a receiver detects a missing packet, it does not send a NACK immediately. It rather schedules a random timer in the future, and when the timer expires, *multicasts* the NACK. In the meantime, if the receiver hears a NACK for the same missing packet, it cancels its pending timer and schedules another random timer. We call this variant of receiver-initiated protocols as receiver-initiated with NACK avoidance (RINA) protocols. It is worthy noting that there is no notion of a memory allocation window (*mw*) in these protocols because the sender has no idea when it is safe to deallocate the buffer space. However, the pacing of the receivers and the retransmission mechanisms (the adjustment of the congestion window) is done in a scalable and efficient manner. One of the premier examples of this category of protocols is SRM [8].

### Ring-based Protocols

Sender-initiated and RINA protocols assume a flat organization of receivers. Ring-based protocols, in contrast, arrange the receivers in the form of a ring, where one of the receivers at any instant of time is designated a token site. The token site is responsible for sending ACKs back to the sender. The token site retransmits the missing packets. The token is passed to the next receiver in the ring when the next receiver has received all the packets that the current token site has received. Once the token is passed, the token site can move the memory allocation window ($mw$). Adjustment of the congestion window ($cw$) is done either by the token site or by the sender. RMP [29] and TRP [4] are examples of this category of protocols.

### Tree-based Protocols

In this class of protocols, the receivers are organized in a tree such that the sender is at the root of the tree, the receivers are at the leaves and domain representatives are at the intermediate points of the tree. Domain representatives (or DRs) represent a group of receivers or a domain and are also organized in a hierarchical manner. The sender is the highest-level DR. In these protocols, the receivers send status messages (ACK+bitmap or ACK+NACK) to the corresponding DR. The sender/DR moves its congestion window ($cw$) based on the ACKs and retransmits missing packets based on the NACKs. [18] also introduced a term called HACK (Hierarchical ACK) to refer to a consolidated ACK of a domain which will be periodically sent by the DRs to the next-level DRs. HACKs help in moving the memory allocation window at the DRs. RMTP [19, 24], LGMP [12], TMTP [30] belong to this category.

A further optimization of these tree-based protocols is to take advantage of the NACK-avoidance scheme of the RINA protocols. These protocols are referred to as

Table 2.1: Summary of Performance Comparison

| Protocol | Sender Requirements | Receiver Requirements |
|---|---|---|
| Sender-Init. | $O(R(1+pln(R)/(1-p)))$ | $O(1-p+pln(R))$ |
| Receiver-Init. | $O(1+pR/(1-p))$ | $O(1-p+pln(R))$ |
| RINA | $O(1+pln(R)/(1-p))$ | $O(1-p+pln(R))$ |
| Ring-Based | $O(1/(1-p))$<br>Token Site: $O(1+(R-1)p/(1-p))$ | $O((1+p^2)/(1-p))$ |
| Tree-Based | $O(B(1-p)+pBln(B))$<br>DR: $O(B(1-p)+Bpln(B))$ | $O(1-p+pln(B))$ |
| Tree-NAPP | $O(1+pln(B)/(1-p))$ | $O(1+((1-p+pln(B)+$<br>$+p^2(1-4p))/(1-p))$ |

Tree-NAPP protocols because they combine the notions of NACK avoidance and periodic polling with the basic tree-based organization of the receivers.

### 2.5.2 Performance Analysis

Performance analysis of the different classes of protocols will be skipped here and we will only summarize the results as surveyed in [23]. A summary of protocol performance complexity is depicted in Table 2.1.

In Table 2.1, $R$ is the number of receivers in the multicast session, $p$ is the probability of packet loss at a receiver, $B$ is the branching factor of the tree (number of receivers local to a repair server), and DR is the domain representative as described in the previous section.

# Chapter 3

# Global Pairing

Most existing reliable transport protocols utilize error recovery schemes where receivers try to recover their lost packets either from the sender or from some repair server in the network. Repairs can also be requested from any receiver, but it is not known at the time which receiver will respond with the lost packet since repair requests are multicast to some or all of the receivers, sometimes including the server. Responses are most of the time multicast to the whole group where all receivers, including the sender, have to listen and process these packets, whether they lost a packet or not. In this chapter, we introduce a new concept for error recovery that primarily involves receivers only. It allows any particular receiver to request repair from another well known receiver, its *buddy*. The two receivers become known to each other as the result of a pairing operation by the server. A new protocol, called the $K$ protocol is then introduced that utilizes the concept of pairing for error recovery.

## 3.1 Network Model and Assumptions

This model consists of one sender $S$ multicasting a continuous stream of packets to $R$ receivers. The model could possibly be extended to have $R$ receivers where any receiver has an equal chance of being a sender itself. In that case, the probability of a receiver being a sender is $1/(R+1)$.

For our model we assume the following:

1. All loss events at all receivers for all transmissions are *mutually independent.*

2. The probability of packet loss, $p$, is independent of the receiver.

3. ACKs and NACKs are never lost and these packets are typically small.

## 3.2 Global Pairing

*Pairing* is a process that randomly pairs two receivers for the purpose of packet loss recovery. In the global mode of the protocol, $S$ acts as as a *Match Maker* and, upon request, pairs a receiver $R_j$ with another receiver $R_i$ that has been waiting to be paired in a First-In-First-Out (FIFO) queue at $S$. The concept of pairing is depicted in Figure 3.1.

## 3.3 New Protocol Proposed

The new protocol proposed in this research, called the $K$ protocol, tries to achieve optimal processing requirements on the sender and on the receivers, and minimal bandwidth – thus a very high throughput – utilizing a new and efficient error recovery scheme. The new protocol exhibits the following properties:

- It utilizes a *receiver-initiated* loss recovery scheme.
- It is NACK-based
- The sender is generally not involved in the recovery of lost packets, unless it is absolutely necessary.
- It introduces the concept of *Global Pairing*. A variation of this protocol uses *Local Pairing* in order to reduce congestion further on the network.

### 3.3.1 The $K$ Protocol

The $K$ protocol exhibits the following behavior:

- Upon joining a multicast group (session), a receiver $R_i$ initially pairs up with the sender $S$ until a buddy is found.
- Receiver $R_i$ informs $S$ that it is looking for a buddy.

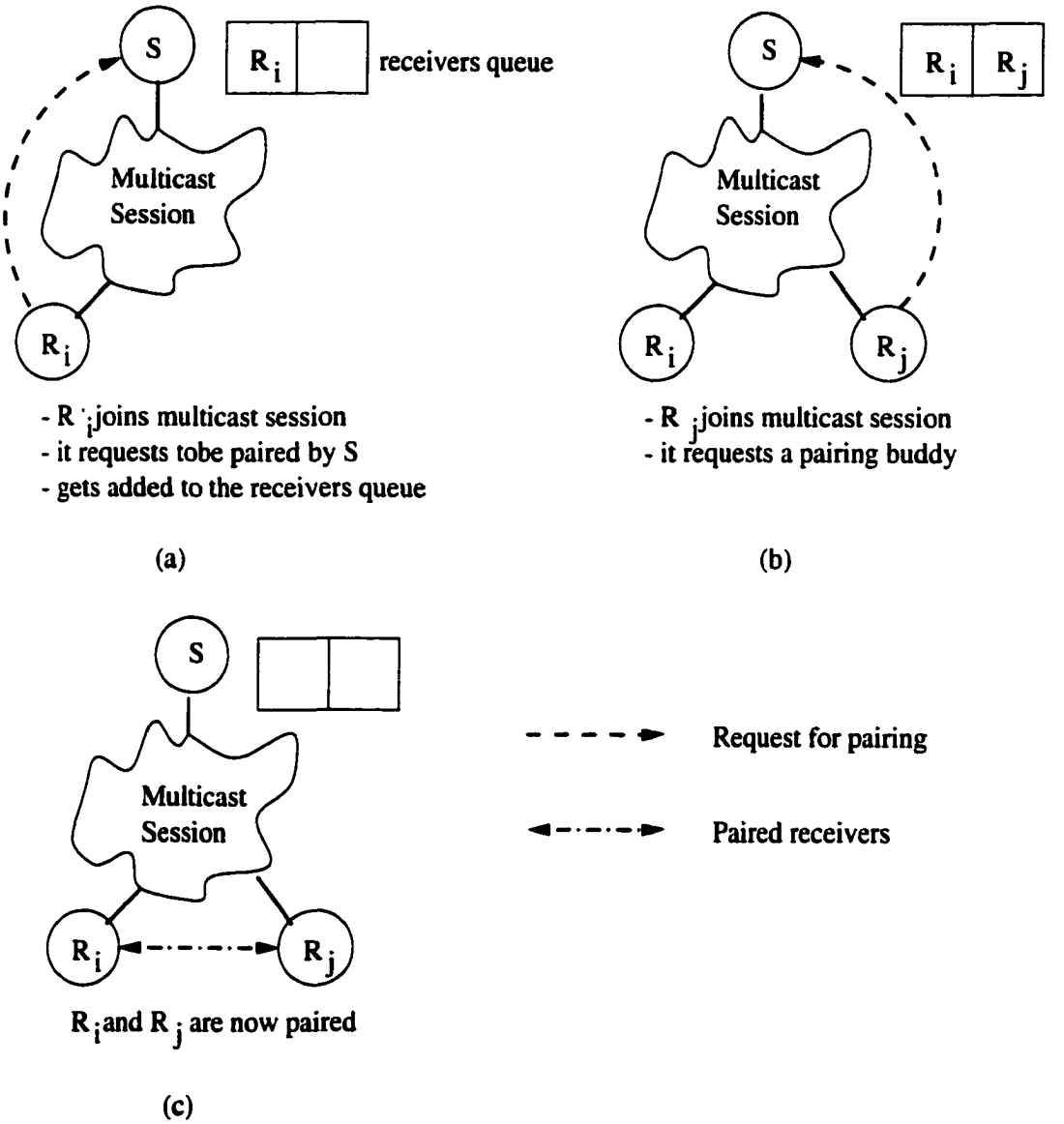



Figure 3.1: The Concept of Receiver Pairing

- $S$ saves the address of $R_i$ and waits until another receiver requests pairing.
- Upon receiving a request for pairing from another receiver $R_j$, $S$ checks if it has any receiver waiting for pairing. If so, then $S$ informs $R_i$ and $R_j$ that both of them are going to be buddies, i.e. they will be a pair for the purpose of error recovery. At that moment $R_i$ drops its pairing with $S$ and pairs with its new buddy $R_j$. If there is no receiver waiting for pairing, $S$ behaves as in the previous step.
- $R_i$ and $R_j$ now use *point-to-point* communication for error correction in a NACK-based fashion (as opposed to multicasting their NACKs). If $R_i$ detects a lost packet, then it sends a NACK to its buddy $R_j$, which then processes the NACK, checks if it actually has the packet, and if so sends the packet to $R_i$. If $R_j$ realizes that it does not have the packet, then it requests the packet from the $S$ and supplies it to itself and $R_i$. Again, all communication here is done via point-to-point reliable communication.
- Upon leaving a multicast session, a receiver informs its buddy that it intends to leave. The buddy then acts as if is has just joined the session, i.e. it pairs with $S$ until a buddy is found.
- If $R_i$ is not responding, then $R_j$ can detect this either by polling $R_i$ periodically, or by expiring timeouts, in which case it behaves as in the previous step.
- If a receiver $R_i$ has been dropped (dumped) by $R_j$, then upon reestablishing a contact with $R_j$, $R_i$ is informed that it needs to find another buddy. $R_i$ then behaves as if it's joining anew.

### 3.3.2 Observations

We observe the following properties for the $K$ protocol:

- $S$ acts as a *Match Maker.*

- Protocol $K$ is *stateless* with respect to pairing, i.e. it keeps no record of which receivers are paired with each other at any point in time. Statelessness is very important to have, otherwise it would be very costly in terms of memory requirements for a server to remember all pairs. In addition, that information will become stale very quickly due to the dynamic nature of receivers joining and leaving a multicast session.
- Unless there are delays in processing pairings, for example due to insufficient processing power, no more than two receivers should pair with $S$ at any point in time, since they will end up being paired with each other.
- If $R$, the total number of receivers, is even, then half of the receivers will be paired with the other half. Otherwise we will have $(R-1)/2$ pairs and one receiver left paired with $S$.

It is worthy to note that all receivers are required to pair under $K$, and that none of them are incapable of doing so. The operation of the $K$ protocol is depicted in Figure 3.2.

In this example, there is one sender $S$ and seven receivers, $R1$ through $R7$. The requests for pairing randomly arrive at $S$ from $R2$, $R4$, $R1$, $R6$, $R3$, $R5$ and $R7$ respectively in that order. The resulting pairings are shown in the figure. Receivers $R3$ through $R7$ lose the packet/segment and request repairs from their buddies. Only in the case of $R5$ did the buddy not have the requested packet. In that case $R5$ requests the packet from $S$ and then supplies it back to $R3$. Note that since the number of receivers was odd, the last receiver to request pairing, namely $R7$, did not find a buddy, so it paired with $S$.

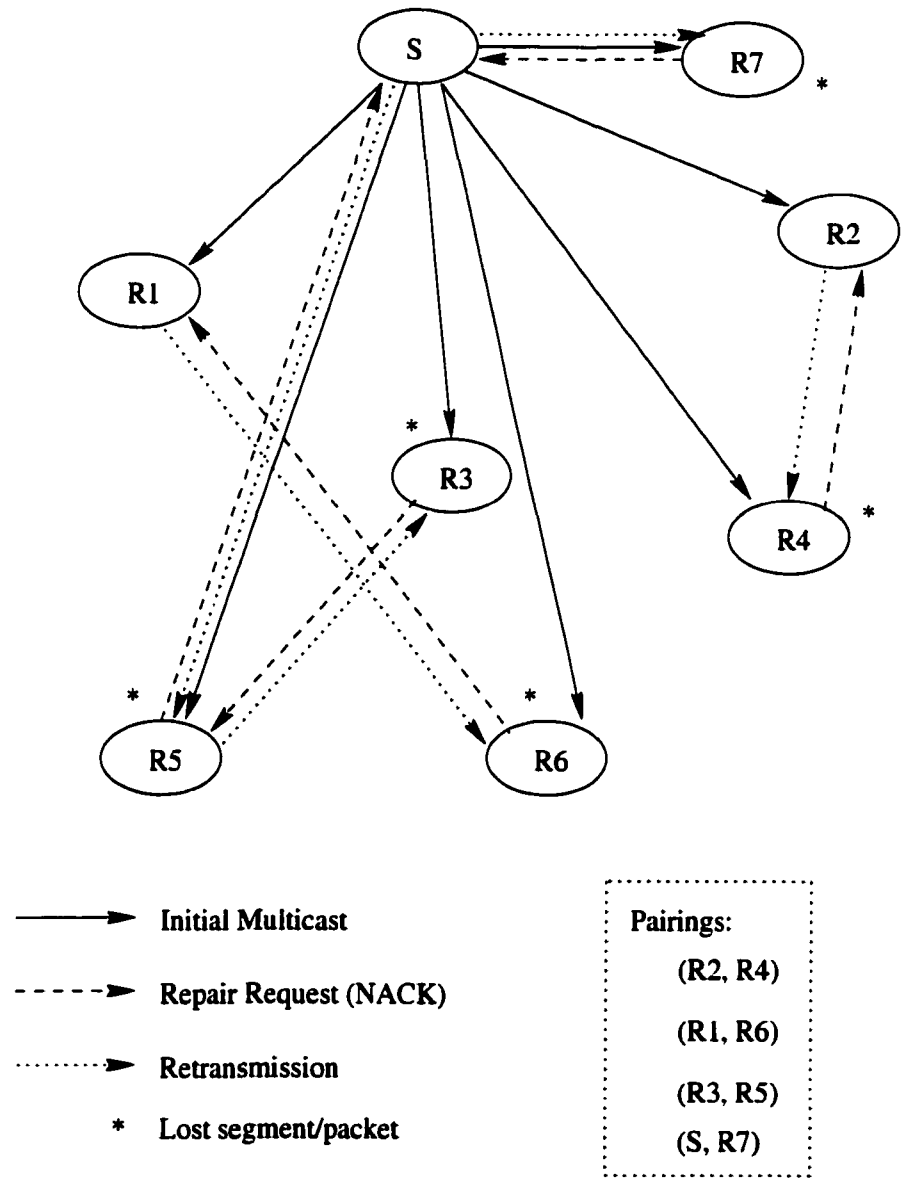



Figure 3.2: Operation of the $K$ Protocol

### 3.3.3 Effect of Pairing on Sender

One important question to pose here is whether pairing will have a severe effect on the sender, or even on the whole network. Let us consider this by observing that the pairing operation involves the reception of two messages from two receivers, the storage of the addresses of those two receivers until they get paired, and the sending of two messages to the two receivers informing them that they are to pair. Message sizes in this operation are very small and the storage needed is insignificant. In addition, pairing is generally a one-time operation for every two receivers for the life of a multicast session, provided the receivers remain healthy for the duration of the session. Therefore, in the best case, pairing only requires the sender to process a total of $2R$ message during the life of the session. In the worst case, all receivers join the session simultaneously – an unlikely scenario – in which case $S$ receives $R$ requests for pairing and is then expected to perform $R/2$ pairing operations. In reality, receivers dynamically join/leave multicast sessions at different points in time, although a surge could be expected at the beginning of a multicast session.

## 3.4 Performance Analysis

It is expected that communication bandwidths will grow at a much higher rate than processing speeds during the next decade [28]. This is why we take the approach of focusing on *processing requirements* of these protocols at both sending and receiving hosts rather than the communication bandwidth requirements. In addition, we are concerned with *scalability*: how well will our approaches handle large numbers of receivers. The scalability of specific ACK- or NACK-based protocols is examined by [8, 24, 1]. One related analytic work is [7], which proposes a NAK-based protocol and compares it with an ACK-based protocol. The focus in [7], however, is on the performance effects of sending periodic state information. They assume

Table 3.1: Notation

| | |
|---|---|
| $X_f$ | the time to feed packet from application to transport at sender |
| $X_p$ | the time to process the transmission of a packet at a sender |
| $X_t$ | the time to process a timeout at the sender |
| $X_n$ | the time to process a NACK at a sender |
| $Y_f$ | the time to feed packet from application to transport at receiver |
| $Y_p$ | the time to process a newly received packet at a receiver |
| $Y_t$ | the time to process a timeout at a receiver |
| $Y_n$ | the time to process and transmit/receive a NACK at a receiver |
| $p$ | the end-to-end loss probability at a receiver |
| $M$ | the number of receivers requesting repair from $S$ |
| $R$ | the number of receivers in the multicast session |
| $X^w, Y^w$ | the send and receive per packet processing time in protocol $w \in \{K, K\prime, KL, KL\prime\}$ |

lossless network and do not consider the overhead of ACK and NACK processing – considerations that are central to our analysis.

In this section we follow the performance analysis model first pioneered by [25]. Terminology specific to this analysis can be found in Table 3.1.

### 3.4.1 Sender

Processing requirements at the sender under the $K$ protocol can be expressed as

$$X^K = X_f + \sum_{i=1}^{M}(X_p(i) + X_n(i)).$$

Taking the expectation of $X^K$ we get

$$E[X^K] = E[X_f] + E[M](E[X_p] + E[X_n])$$

where $M$ is the number of receivers requesting repair from $S$ when their buddies cannot supply the packet. Here the concept of finding out the number of transmissions necessary for all receivers to correctly receive a packet does not apply since error correction is done in a reliable one-to-one communication. Therefore, we are only interested in finding the number of receivers that will request correction from $S$.

The probability of a pair $(R_i, R_j)$ not receiving a packet is now $p^2$. We can now treat the system as a session of $R/2$ receivers (each being a pair of receivers) with the probability of losing a packet of $p^2$. The expected value of $M$ can now be given as

$$E[M] = \frac{p^2}{2}R.$$

Substituting in $E[X^K]$ we get

$$E[X^K] = E[X_f] + \frac{p^2}{2}R(E[X_p] + E[X_n]).$$

Therefore,

$$E[X^K] \in O(1 + \frac{p^2}{2}R).$$

As $p \longrightarrow 0$, $E[X^K] = O(1)$.

### 3.4.2 Receiver

In order to analyze the processing requirements at the receiver we must first consider the following facts:

- receiver $R_i$ only receives NACKs from its buddy $R_j$ — exactly one NACK.
- receiver $R_i$ supplies $R_j$ with missing packet via point-to-point communication, at most once per packet transmission/loss.

- $R_i$ asks $S$ for a missing packet only if it does not have it itself. This takes place at most once per packet transmission/loss.

In doing performance analysis at the receiver we must consider the time it takes the receiver to do the following:

- time to obtain data from higher layers
- time to process NACK from buddy
- time to send NACK to buddy or to $S$ if it does not have the packet
- time to process received packet from buddy or from $S$
- time to send packet to buddy

Therefore, the expected processing time at the receiver can be expressed as the sum of the above times to get

$$Y^K = Y_f + Y_n + X_n + Y_p + X_p.$$

Taking the expectation of $Y^K$ we get

$$E[Y^K] = O(1).$$

That means that processing requirements at the receiver are *independent of* $R$, which is a highly desirable property in order to achieve optimality. This is one of the major results of this research.

# Chapter 4

# Local Pairing

## 4.1 Introduction

In this chapter, we apply the concept of *Local Recovery* to Global Pairing. We call the resulting concept *Local Pairing.* The main idea is to use a *divide-and-conquer* approach and divide the receivers into *local regions.* Each local region will have a designated local *repair server* that handles repair and retransmission requests from its local receivers, and only requests repairs from the server when it does not have the requested data segment. The $K$ protocol can then be applied locally in each region where the buddies come from the same local region, which eventually improves end-to-end latency.

## 4.2 Related Work

Many researchers have recently proposed different approaches for local recovery [16]. Some of these approaches are scalable reliable multicast, SRM (with local recovery enhancements) [8], local group concept, LGC [12], tree-based multicast transport protocol, TMTP [30], LORAX [18], log-based receiver reliable multicast, LBRM [13] and reliable multicast transport protocol, RMTP [19]. Following the classification we saw in earlier chapters, LBRM is server-based, SRM with local recovery enhancements is self organizing receiver-based and RMTP, TMTP and LORAX are designated receiver-based with pre-constructed logical hierarchy. LGC is a hybrid of the logical tree-based approach and SRM. The new dimension we are

adding in this research is the combination of the concept of pairing under the $K$ protocol with local recovery. We call this approach Local Pairing.

## 4.3 Local Pairing

We now describe a generic receiver-based reliable multicast local recovery protocol, $KL$, that assumes the presence of a *repair tree* similar to the one in [16]. This repair tree is the physical multicast tree constructed by the routing protocols with the sender as the root, receivers as leaves and repair servers co-located with routers. The receivers recover lost packets from repair servers only if their buddies under the local $K$ protocol do not have the lost packets. Repair servers, in turn, recover lost packets from either upper level repair servers or the sender. The repair servers themselves can also be paired with each other for error recovery before consulting with the sender.

### 4.3.1 The $KL$ Protocol

Protocol $KL$, a combination of the Local Pairing $K$ protocol and Local Recovery, exhibits the following behavior:

- Upon joining a multicast group (session), a receiver $R_i$ initially pairs up with its local repair server ($RS$) until a buddy is found.
- Receiver $R_i$ informs $RS$ that it is looking for a buddy.
- $RS$ saves the address of $R_i$ and waits until another receiver from the same local region requests pairing.
- Upon receiving a request for pairing from another receiver $R_j$, $RS$ checks if it has any receiver waiting for pairing. If so, then $RS$ informs $R_i$ and $R_j$ that both of them are going to be buddies, i.e. they will be a pair for the purpose of local error recovery. At that moment $R_i$ drops its pairing with $RS$ and pairs

with its new buddy $R_j$. If there is no receiver waiting for pairing, $RS$ behaves as in the previous step.

- $R_i$ and $R_j$ now use *point-to-point* communication for error correction in a NACK-based fashion (as opposed to multicasting their NACKs, whether local or global). If either $R_i$ detects a lost packet, then it sends a NACK to its buddy $R_j$, which then processes the NACK, checks if it actually has the packet, and if so sends the packet to $R_i$. If $R_j$ realizes that it does not have the packet, then it requests the packet from the $RS$ and supplies it to itself and $R_i$. Again, all communication here is done via point-to-point reliable communication.
- Upon leaving a multicast session, a receiver informs its buddy that it intends to leave. The buddy then acts as if is has just joined the session, i.e. it pairs with $RS$ until a buddy is found.
- If $R_i$ is not responding, then $R_j$ can detect this either by polling $R_i$ periodically, or by expiring timeouts, in which case it behaves as in the previous step.
- If a receiver $R_i$ has been dropped (dumped) by $R_j$, then upon reestablishing a contact with $R_j$, $R_i$ is informed that it needs to find another buddy. $R_i$ then behaves as if it's joining anew.

### 4.3.2 Observations

We observe the following properties for the $KL$ protocol:

- $RS$ acts as a *Match Maker* for its local region.
- Just like $K$, Protocol $KL$ is also *stateless* with respect to pairing.
- Unless there are delays in processing pairings, for example due to insufficient processing power, no more than two receivers should pair with $RS$ at any point in time, since they will end up being paired with each other.

- If $B$, the total number of receivers local to $RS$, is even, then half of the receivers will be paired with the other half. Otherwise we will have $(B-1)/2$ pairs and one receiver left paired with $RS$.

The operation of the $KL$ protocol is depicted in Figure 4.1.

In this example, there is one sender $S$, two repair servers $RS1$ and $RS2$, and seven receivers $R1$ through $R7$. Requests for pairing arrive at $RS1$ from $R1$, $R6$, $R3$ and $R5$ respectively in that order. Requests for pairing arrive at $RS2$ from $R2$, $R4$ and $R7$ respectively in that order. The resulting pairings are shown in the figure. Note that no requests for pairing arrived at $S$. Also, note that for the local region of $RS1$, the number of receivers is even and they were all paired. In the case of $RS2$, $R7$ was left out, so it remained paired with $RS2$. When $R3$ requested repair from its buddy $R5$, $R5$ did not have the packet either, so it requested it from $RS1$. In the case of $R7$, whose buddy was $RS2$ itself, $R2$ did not have the packet itself, so it requested it from $S$.

### 4.3.3 Effect of Local Pairing on $RS$

The analysis is analogous to the effect of pairing on the sender discussed in the previous chapter and will be left out here. However, one major difference is that the total number of local receivers is much smaller than $R$. Therefore, the effect will be even more insignificant on the local repair server.

## 4.4 Pairing of Repair Servers

The concept of pairing of receivers can be easily extended to the repair servers themselves. Instead of requesting repairs from the sender or any higher repair server, repair servers can themselves utilize the buddy system and request pairing from $S$. In that case we observe the following behavior:

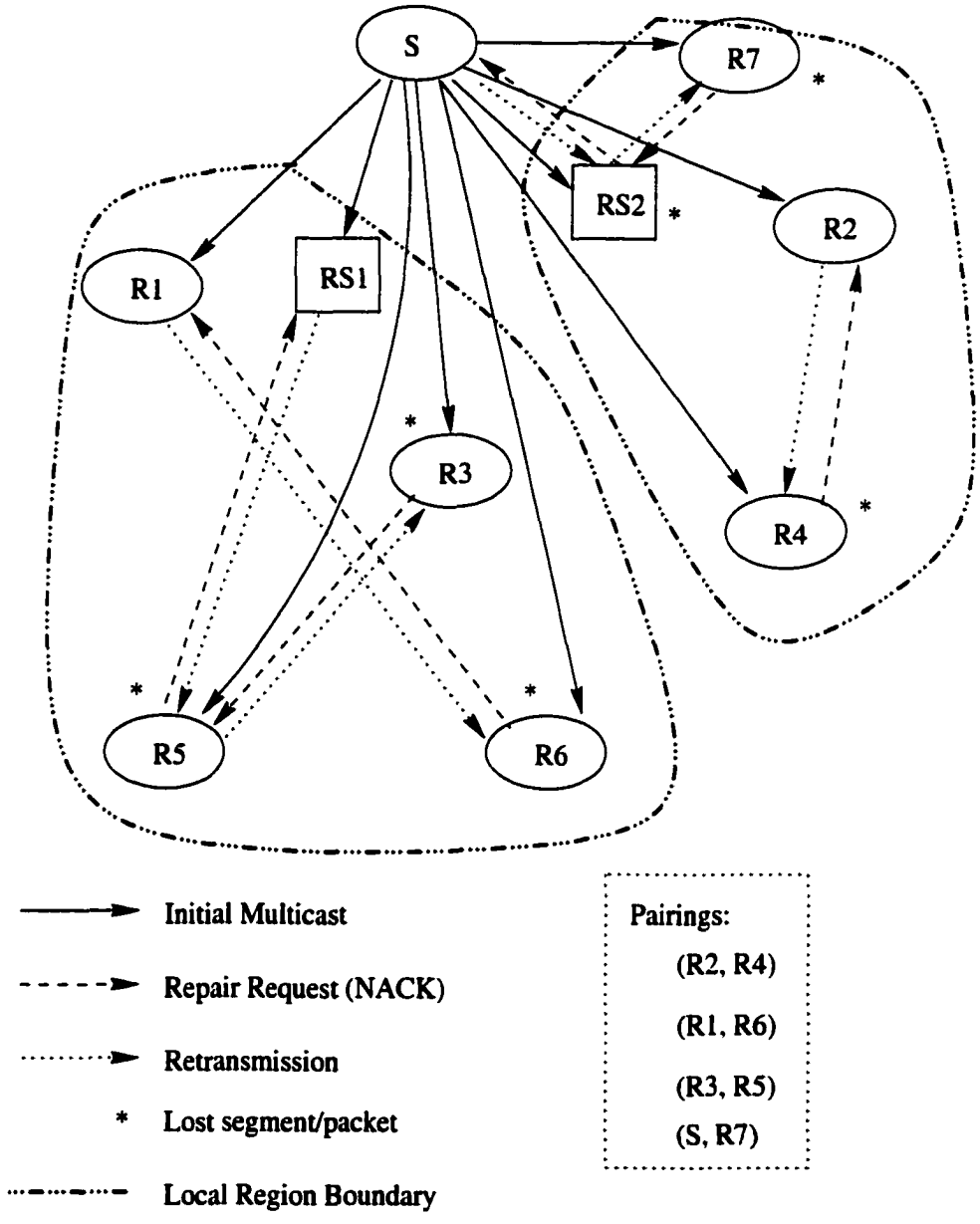



Figure 4.1: Operation of the $KL$ Protocol

- Upon joining a multicast group (session), a repair server $RS_i$ initially pairs up with $S$ until a buddy repair server is found.
- $RS_i$ informs $S$ that it is looking for a buddy.
- $S$ saves the address of $RS_i$ and waits until another repair server requests pairing.
- Upon receiving a request for pairing from another repair server $RS_j$, $S$ checks if it has any repair server waiting for pairing. If so, then it informs $RS_i$ and $RS_j$ that both of them are going to be buddies, i.e. they will be a pair for the purpose of error recovery. At that moment $RS_i$ drops its pairing with $S$ and pairs with its new buddy $RS_j$. If there is no repair server waiting for pairing, $S$ behaves as in the previous step.
- $RS_i$ and $RS_j$ now use *point-to-point* communication for error correction in a NACK-based fashion (as opposed to multicasting their NACKs). If $RS_i$ receives a request for a lost packet from one of its local area receivers, then it sends a NACK to its buddy $RS_j$, which then processes the NACK, checks if it actually has the packet, and if so sends the packet to $RS_i$. If $RS_j$ realizes that it does not have the packet, then it requests the packet from the $S$ and supplies it to itself and $RS_i$. Again, all communication here is done via point-to-point reliable communication.
- Upon leaving a multicast session, a repair server informs its buddy that it intends to leave. The buddy then acts as if is has just joined the session, i.e. it pairs with $S$ until a buddy is found.
- If $RS_i$ is not responding, then $RS_j$ can detect this either by polling $RS_i$ periodically, or by expiring timeouts, in which case it behaves as in the previous step.

- If a repair server $RS_i$ has been dropped (dumped) by $RS_j$, then upon reestablishing a contact with $RS_j$, $RS_i$ is informed that it needs to find another buddy. $RS_i$ then behaves as if it's joining anew.

We observe in this scheme that $S$ plays Match Maker for the repair servers. The server will be involved in error recovery only when any two buddies do not have a requested packet. This scheme will further reduce processing requirements on $S$ and result in a balanced protocol that shifts more of the burden of recovery to the repair servers. The cost of processing on the receiver remains the same, except that end-to-end latency becomes much smaller since requests and retransmissions do not have to travel all the way to or from the sender.

## 4.5 Performance Analysis

### 4.5.1 Sender

We can express the processing requirement at the sender as the total time it takes to perform the following:

- prepare and transmit the first packet (includes feeding the packet from the application to the transport layer)
- retransmit lost packets
- process hierarchical acknowledgments

That can be expressed as

$$X^{KL} = X_f + X_p(1) + \sum_{m=2}^{M}(X_t(m) + X_p(m)) + \sum_{i=1}^{L^H} X_h(i)$$

where $L^H$ is the number of HACKs received at the sender. Taking expectations we get

$$E[X^{KL}] = E[X_f] + E[M]E[X_p] + (E[M] - 1)E[X_t] + E[L^H]E[X_H].$$

The expected number of HACKs received by the sender can be expressed as:

$$E[L^H] = E[M]B(1-p)$$

where $B$ is the number of children of the sender and $E[M]$ is the expected number of transmissions needed to deliver a packet to a receiver. Substituting into $E[X^{KL}]$ we get

$$E[X^{KL}] = E[X_f] + E[M]E[X_p] + (E[M]-1)E[X_t] + E[M]B(1-p)E[X_H].$$

It can be shown (see [28] for details) that

$$E[M] \in O(1 + pln(R)/(1-p)).$$

Substituting $B$ for $R$ in $E[M]$ we get

$$E[M] \in O(1 + pln(B)/(1-p)).$$

This implies that

$$E[X^{KL}] \in O(B(1-p) + Bpln(B)).$$

Notice that if $p$ is constant, then $E[X^{KL}]$ is $O(Bln(B))$. In addition, if $p \longrightarrow 0$, then $E[X^{KL}] = O(1)$.

If the repair server themselves are grouped using the $K$ protocol, then the end-to-end loss probability becomes $p^2$ both for receivers and repair servers since they are being paired two receivers/repair servers at a time. Furthermore, the number of children of the sender can now be regarded as $B/2$ and receivers as $R/2$. Substituting

in $E[M]$ we get a new expression for $E[X^{KL}]$ given by

$$E[X^{KL}] \in O(\frac{B}{2}(1-p^2) + \frac{B}{2}p^2 ln(\frac{B}{2})).$$

### 4.5.2 Receiver

The expected processing time at the receiver will not change under $KL$ since, as far as the receiver is concerned, the local repair server is indistinguishable from a sender. A receiver will request pairing from its local repair server the same way it did from $S$. Similarly, it will request repairs from the repair server just like it did with $S$. The behavior of the receiver and the processing requirements are described in section 3.4.2 with $S$ being the repair server in this case.

Therefore,

$$E[Y^{KL}] \in O(1).$$

## 4.6 Summary

In this chapter we applied the concept of Local Recovery to Global Pairing and we called the resulting concept Local Pairing. The use of repair servers to retransmit lost packets to its local group of receivers relieves the sender and reduces its performance complexity from a linear to a logarithmic one. The extra work is now handled by the repair servers. The $K$ protocol is then applied locally in each group, where the local repair server is regarded by the receivers just like $S$. The only difference is that the repair server does not always have the missing packet and it only supplies missing ones (as opposed to $S$ multicasting the original packets). We consider the repair servers to be part of the backbone network whose congestion and performance is not studied here due to the expectation we make as suggested by [28], which states

that communication bandwidths will grow at a much higher rate than processing speeds during the next decade.

# Chapter 5

# A Generalized Grouping Protocol

## 5.1 Introduction

So far under the $K$ protocol, only two receivers are paired together and become buddies for the purpose of error recovery. Although the processing cost on the receiver was $O(1)$, that on the sender was still $O(R)$, which meant that the size of the multicast session will play a big role in the processing requirements on the sender. In the class of receiver-initiated protocols, *without* any form of hierarchical organization, the class of protocols that exhibited the best performance was that of receiver-initiated with NACK avoidance (RINA). For constant error probability $p$, RINA protocols were $O(ln(R))$. Can we do any better?

In this chapter we extend the grouping concept to a group of $n$ receivers that will buddy together for the purpose of error recovery. Instead of only two receivers, $n$ receivers will now form a group and they will cooperate in a well defined manner to recover lost packets, even when all but one of them has lost those packets. The immediate question that comes to mind is how much more bandwidth will this incur on the network. As we will see in the performance analysis section, it will not be much for a constant group size. In addition, it is worth noting that communication bandwidths are expected to grow at a much higher rate than processing speeds during the next decade [28]. This is why we are focussed on processing costs at the sender and at the receiver.

A modified protocol, called the $K\prime$ protocol, is introduced that utilizes the concept of *grouping* for error recovery.

## 5.2 Network Model and Assumptions

We assume the same model proposed earlier for the Global Pairing mode, which consisted of one sender $S$ multicasting a continuous stream of packets to $R$ receivers. We add one last assumption here and list all four as follows:

1. All loss events at all receivers for all transmissions are *mutually independent.*
2. The probability of packet loss, $p$, is independent of the receiver.
3. ACKs and NACKs are never lost and these packets are typically small.
4. Processing requirements at the hosts are more important than network bandwidth in determining the throughput of reliable multicast protocols.

## 5.3 Grouping

*Grouping* is a process that randomly groups $n$ receivers for the purpose of packet loss recovery. In the global mode of the protocol, $S$ acts as as a *Match Maker* and, upon request, groups $n$ receivers $R_1$, $R_2$,...,$R_n$ with each other, $(n-1)$ of which had been waiting to be grouped in a queue at $S$. Global Pairing now becomes a special case of Grouping where $n = 2$.

## 5.4 A New Modified Protocol

The new protocol proposed in this chapter, called the $K\prime$ protocol, is a generalization of the $K$ protocol proposed earlier. This protocol tries to achieve optimal processing requirements on both the sender and the receivers without incurring heavy additional bandwidth. It utilizes a new and efficient error recovery scheme. The protocol exhibits the following properties:

- It utilizes a *receiver-initiated* loss recovery scheme.

- It is NACK-based.
- The sender is generally not involved in the recovery of lost packets, unless no other receiver can provide the missing packets.
- It introduces the concept of *Grouping*. A variation of this protocol uses *Local Grouping* in order to reduce congestion further on the network.

### 5.4.1 The $K\prime$ Protocol

The $K\prime$ protocol exhibits the following behavior:

- Upon joining a multicast group (session), a receiver $R_i$ initially pairs up with the sender $S$ until a *group* $G$ of $n$ receivers is formed.
- Receiver $R_i$ informs $S$ that it is looking for a group.
- $S$ saves the address of $R_i$ in a queue and waits until enough receivers request grouping.
- Upon receiving a request for grouping, $S$ checks if it has $(n-1)$ receivers waiting for grouping. If so, then $S$ informs $R_i$ and and all other $(n-1)$ receivers in the queue that all of them are going to be buddies in the same group for the purpose of error recovery. At that moment $R_i$ drops its pairing with $S$ and groups with its new buddies in group $G$. If there are no receivers waiting for grouping, $S$ behaves as in the previous step.
- All receivers in $G$ now use *point-to-point* communication for error correction in a NACK-based fashion (as opposed to multicasting their NACKs). When group members detect lost packets, they use one of the *Group Recovery Schemes* described in the next section to recover the lost packets. The schemes work in such a way that all $n$ receivers will end up having the missing packet. If at least one member has it, then $S$ should not be bothered and all members should recover their loss from that one receiver. Only if none of the receivers in $G$ has a missing packet does $S$ get consulted.

- Upon leaving a multicast session, a receiver informs its group members that it intends to leave. The group members then break the group and act as if is they have just joined the session, i.e. they pair with $S$ until enough members are found to form a new group.
- If some receiver in a group is not responding, then its buddies in the group can detect this either by polling it periodically, or by expiring timeouts, in which case they behave as in the previous step.
- If a receiver $R_i$ has been dropped (dumped) by its group members, then upon reestablishing a contact with any of the former members, $R_i$ is informed that it needs to find another group. $R_i$ then behaves as if it's joining anew.

### 5.4.2 Group Recovery Schemes

Receivers in a group can deploy several error recovery schemes, but some are more efficient than others, especially when a large number of receivers in the group did not receive a multicast packet. We only present three schemes in this section, each with its advantages and disadvantages. However, since the number of receivers in a group is constant, even a total ordering on the sending of NACKs and retransmissions (an $O(n^2)$ operation) would not pose a severe threat on the processing load of the receivers.

#### All-NACK

In this scheme all receivers in a group set a random timer and upon expiration of the timer send $(n-1)$ NACKs to all other receivers requesting the missing packet. The receiver processes $(n-1)$ replies from its buddies and if any of them has replied with the missing packet, it retains a copy and sends the packet to all receivers that did not have it. This way all receivers in the group recover. Upon hearing a NACK for the same packet for which a receiver has set a timer, or if a packet is received

from a buddy before a timer expires, then a receiver resets or cancels its own timer, respectively. The idea behind setting a timer and waiting for it to expire is to allow only one receiver to send the NACKs and eventually supply all its buddies with the missing packet. However, if none of the buddies has the missing packet, then the receiver sends a NACK to $S$, recovers the packet, and sends it to all its buddies. If timers are not set properly, more than one buddy may send $(n-1)$ NACKs, thus requiring more processing on the part of each receiver in the group.

### Ring Search

In this scheme the sender organizes the $n$ receivers in a *ring* such that a receiver always knows who its successor is in the ring. If a receiver detects a lost packet, then it sends a NACK to its successor in the ring. If the successor has the packet it supplies it, otherwise it asks its own successor, and so on until the packet is fetched from some buddy. All receivers along that portion of the ring from the initial receiver to the one that had the missing packet will recover as a result. Other receivers that missed the packet will behave in a similar way. Eventually, all receivers in the group will have the packet. However, if the receiver that initially requested a repair receives a NACK from some receiver in the group *for the same lost packet*, then the message must have come from its predecessor in the ring. This servers as an indication that none of the buddies has the packet, in which case the receiver recovers the packet from $S$ and sends it to all its buddies. The disadvantage in this scheme is that recover proceeds in one direction, which means that every member may be asked for the packet in a sequential fashion, which affects latency.

**Bidirectional Ring Search**

This scheme is similar to the Ring Search, except that receivers are now arranged in a *doubly-linked ring* where each receiver always knows its successor as well as its predecessor in the ring. Upon detecting a lost packet, a receiver sends two NACKs, one to its successor and one to its predecessor. Recovery of the lost packet now proceeds in both directions on the ring until all receivers recover the packet. If a receiver receives a NACK *for the same lost packet* both from its successor and from its predecessor, then the ring must have been exhausted, which serves as an indication that all receivers in the group have missed the packet. The receiver detecting this recovers the packet from $S$ and supplies it to all its buddies. This scheme parallelizes the search for the missing packet, which cuts down on latency.

All three recovery schemes are depicted in Figure 5.1.

### 5.4.3 Observations

We observe the following properties for the $K\prime$ protocol:

- $S$ acts as a *Match Maker* to form $G$.
- Protocol $K\prime$ is *stateless* with respect to grouping, i.e. it keeps no record of which receivers are grouped with each other at any point in time. Statelessness is very important to have, otherwise it would be very costly in terms of memory requirements for a server to remember all groups. In addition, that information will become stale very quickly due to the dynamic nature of receivers joining and leaving a multicast session.
- Unless there are delays in processing groupings, for example due to insufficient processing power, no more than $(n-1)$ receivers should pair with $S$ at any point in time, since they will end up being grouped with each other.

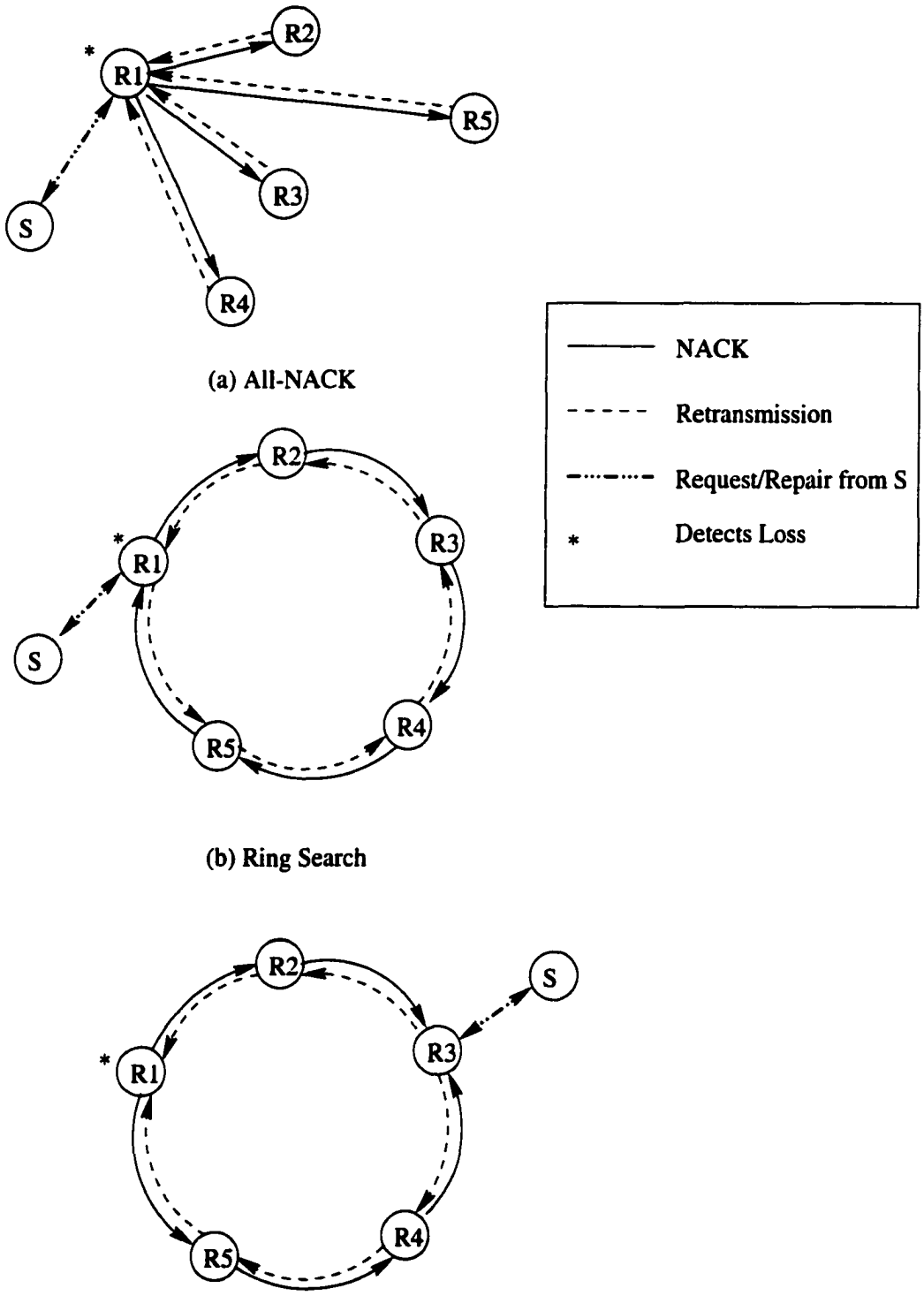



Figure 5.1: Group Recovery Schemes Under $K_1$

- If $R$, the total number of receivers, is a multiple of $n$, then $R/n$ groups will be formed with no left over receivers. Otherwise we will have at most $R$ mod $n$ receivers paired with $S$ at any point in time.

It is worthy to note that all receivers are required to group under $K\prime$, and that none of them are incapable of doing so. The operation of the $K\prime$ protocol is depicted in Figure 5.2.

In this example, there is one sender $S$ and seven receivers, $R1$ through $R7$. The requests for grouping randomly arrive at $S$ from $R2$, $R4$, $R1$, $R6$, $R3$, $R5$ and $R7$ respectively in that order. The resulting groupings are shown in the figure. Receivers $R3$ through $R7$ lose the packet/segment and request repairs from their buddies in the group. In group $(R2, R4, R1)$ $R4$ did not have the packet so it recovered it from $R4$. All receivers in the group $(R6, R3, R5)$ lost the segment, so one of them ($R5$ in this case) recovered the segment from $S$ and supplied it to all others. Note that since 7 mod 3 = 1, only one receiver (the last to request grouping), namely $R7$, did not find a group, so it paired with $S$.

### 5.4.4 Effect of Grouping on Sender

Similar to the case of Global Pairing, the question to pose again is whether grouping will have a severe effect on the sender, or even on the whole network. Let us consider this by observing that the grouping operation involves the reception of $n$ messages from $n$ receivers, the storage of the addresses of $(n-1)$ receivers until they get grouped, and the sending of $n$ messages to all $n$ receivers informing them that they are to form a group. Message sizes in this operation are very small and the storage needed is insignificant. We only need $(n-1)$ additional storage buffers to store the addresses of $(n-1)$ receivers until the $n$th receiver requests grouping. In addition, grouping is generally a one-time operation for every $n$ receivers for the life

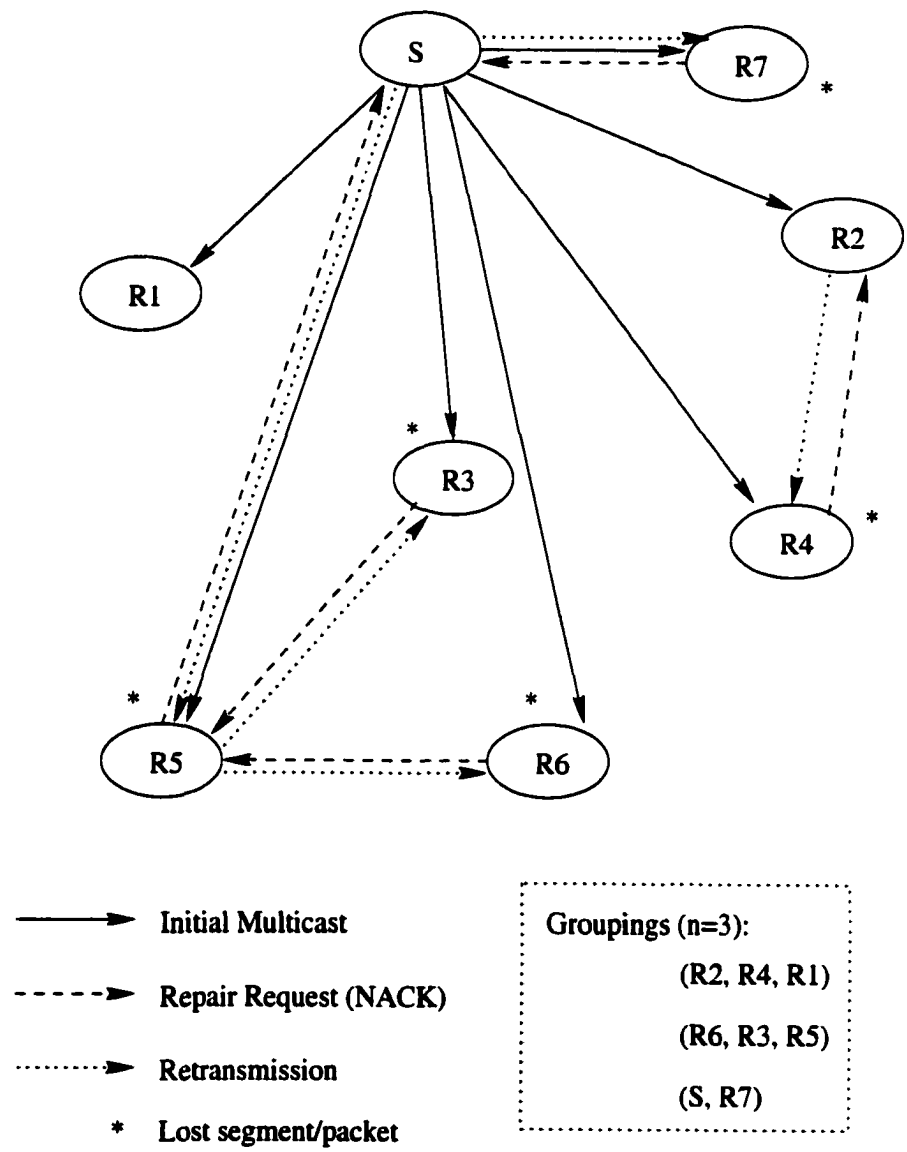



Figure 5.2: Operation of the $K_1$ Protocol

Table 5.1: Notation

| | |
|---|---|
| $X_f$ | the time to feed packet from application to transport |
| $X_p$ | the time to process the transmission of a packet at a sender |
| $X_n$ | the time to process a NACK at a sender |
| $Y_p$ | the time to process a newly received packet at a receiver |
| $Y_t$ | the time to process a timeout at a receiver |
| $Y_n$ | the time to process and transmit/receive a NACK at a receiver |
| $p$ | the end-to-end loss probability at a receiver |
| $M$ | the number of receivers requesting repair from $S$ |
| $R$ | the number of receivers in the multicast session |
| $n$ | the number of receivers (buddies) in a group |
| $X^w, Y^w$ | the send and receive per packet processing time in protocol $w \in \{K, K\prime, KL, KL\prime\}$ |

of a multicast session, provided the receivers remain healthy for the duration of the session. Therefore, in the best case, grouping only requires the sender to process a total of $2R$ messages during the life of the session. In the worst case, all receivers join the session simultaneously – an unlikely scenario – in which case $S$ receives $R$ requests for grouping and is then expected to perform $R/n$ grouping operations. In reality, receivers dynamically join/leave multicast sessions at different points in time, although a surge could be expected at the beginning of a multicast session.

## 5.5 Performance Analysis of $K\prime$

In this section we follow the same performance analysis model used in previous chapters. Terminology specific to this analysis can be found in Table 5.1.

### 5.5.1 Sender

Processing requirements at the sender can be expressed as the total time needed to perform the following:

- prepare and transmit the first packet

- process all NACKs sent by receivers
- process all retransmissions to receivers in response to the NACKs

Therefore, the processing requirements at the sender under the $K\prime$ protocol can be expressed as

$$X^{K\prime} = X_f + \sum_{i=1}^{M}(X_n(i) + X_p(i))$$

where $X_n(i)$ is the requirement for processing a NACK from the $i$-th receiver and $X_p(i)$ is the packet processing requirement associated with the $i$-th receiver requesting retransmission of a given packet. $X_f$ is the processing required to prepare and transmit the first packet.

We assume that processing requirements have general distributions, that they are independent of each other, and that $X_n(i)$ and $X_p(i)$ are each identically distributed sequences of random variables. As a result, we omit the argument $i$.

We are interested in the mean processing requirement per packet in order for the sender to multicast packets reliably to all of the receivers. Taking the expectation of $X^{K\prime}$ we get

$$E[X^{K\prime}] = E[X_f] + E[M](E[X_p] + E[X_n])$$

where $M$ is the number of receivers requesting repair from $S$ when their buddies cannot supply the packet. Here the concept of finding out the number of transmissions necessary for all receivers to correctly receive a packet does not apply since error correction is done in a reliable one-to-one communication. Therefore, we are only interested in finding the number of receivers that will request correction from $S$.

The probability of a group $(R_1, R_2, \ldots, R_n)$ not receiving a packet is now reduced to $p^n$. We can now treat the system as a session of $R/n$ receivers (each being a group

of $n$ receivers) with the probability of losing a packet of $p^n$. The expected value of $M$ can now be given as

$$E[M] = \frac{p^n}{n}R.$$

Substituting in $E[X^{K\prime}]$ we get

$$E[X^{K\prime}] = E[X_f] + \frac{p^n}{n}R(E[X_p] + E[X_n]).$$

Please note that the $n$ in $X_n$ is for NACKs and is not the same as the size of a group. Therefore,

$$E[X^{K\prime}] \in O(1 + \frac{p^n}{n}R).$$

As $p \longrightarrow 0$, $E[X^{K\prime}] = O(1)$.

### 5.5.2 Receiver

In order to analyze the processing requirements at the receiver we must first consider the following facts:

- receiver $R_i$ only receives NACKs from its buddies in the group, which amount to at most $(n-1)$ NACKs.
- receiver $R_i$ supplies its $(n-1)$ buddies with missing packets via point-to-point communication, at most $(n-1)$ times per packet transmission/loss.
- A receiver asks $S$ for a lost packet only if none of its buddies in the group have it. This takes place at most once per packet transmission/loss.

In doing performance analysis at the receiver we must consider the time it takes the receiver to do the following:

- obtain data from higher layers
- process NACK from $(n-1)$ buddies
- send $(n-1)$ NACKs to its buddies or to $S$ if none have the packet

- process received packet from buddies (at most $(n-1)$ times) or from $S$ (once)

- send packet to buddies (at most $(n-1)$ times)

Therefore, the expected processing time at the receiver can be expressed as the sum of the above times to get

$$Y^{KI} = Y_f + (n-1)Y_n + (n-1)X_n + (n-1)Y_p + (n-1)X_p.$$

Taking the expectation of $Y^{KI}$ we get

$$E[Y^{KI}] = E[Y_f] + E[n-1]E[Y_n] + E[n-1]E[X_n] + E[n-1]E[Y_p] + E[n-1]E[X_p].$$

Now since $E[n] = n$ for constant $n$, we get

$$E[Y^{KI}] = O(n).$$

Therefore,

$$E[Y^{KI}] \in O(1).$$

That means that processing requirement at the receiver is *constant* and is *independent of R*, which is a highly desirable property in order to achieve optimality. This is another one of the major results of this research.

## 5.6 Grouping with Local Recovery

Similar to the work done in the previous chapter, we can combine the concept of Local Recovery with that of Grouping to achieve even lower processing requirements on the sender. The major difference in the mechanics of this combined protocol, called the *KLI* protocol, is that repair servers now play the role of the sender in

terms of grouping and error recovery. Only if a repair server does not have a lost packet does it consult the sender or another repair server in the hierarchy.

### 5.6.1 The *KL/* Protocol

The *KL/* protocol exhibits the following behavior:

- Upon joining a multicast group (session), a receiver $R_i$ initially pairs up with its local repair server $RS_l$ until a *group* $G$ of $n$ receivers is formed.
- Receiver $R_i$ informs $RS_l$ that it is looking for a group.
- $RS_l$ saves the address of $R_i$ in a queue and waits until enough receivers request grouping.
- Upon receiving a request for grouping, $RS_l$ checks if it has $(n-1)$ receivers waiting for grouping. If so, then $RS_l$ informs $R_i$ and and all other $(n-1)$ receivers in the queue that all of them are going to be buddies in the same group for the purpose of error recovery. At that moment $R_i$ drops its pairing with $RS_l$ and groups with its new buddies in group $G$. If there are no receivers waiting for grouping, $RS_l$ behaves as in the previous step.
- All receivers in $G$ now use point-to-point communication for error correction in a NACK-based fashion and use one of the Group Recovery Schemes described earlier.
- Upon leaving a multicast session, a receiver informs its group members that it intends to leave. The group members then break the group and act as if is they have just joined the session, i.e. they pair with $RS_l$ until enough members are found to form a new group.
- If some receiver in a group is not responding, then its buddies in the group can detect this either by polling it periodically, or by expiring timeouts, in which case they behave as in the previous step.

Table 5.2: Notation

| | |
|---|---|
| $X_h$ | the time to process a HACK |
| $X_H$ | the time to process reception of a HACK |
| $L^H$ | the total number of a HACKs |
| $B$ | the branching factor of a tree |
| $M$ | the number of transmissions needed to deliver a packet to a receiver |
| $X^w, Y^w$ | the send and receive per packet processing time in protocol $w \in \{K, K\prime, KL, KL\prime, KLL\prime\}$ |

- If a receiver $R_i$ has been dropped (dumped) by its group members, then upon reestablishing a contact with any of the former members, $R_i$ is informed that it needs to find another group. $R_i$ then behaves as if it's joining anew.

### 5.6.2 Performance Analysis of $KL\prime$

Processing requirements on the sender will definitely drop down due to the fact that repair servers in the network will now handle repair requests from receivers in their local regions. Repair servers only consult with $S$ when it's absolutely necessary. The resulting processing requirements for the sender and the receiver are described in the following two sections. Notation specific to local recovery can be found in Table 5.2.

#### Sender

We can express the processing requirement at the sender as the total time it takes to perform the following:

- prepare and transmit the first packet (includes feeding the packet from the application to the transport layer)
- retransmit lost packets
- process hierarchical acknowledgments

That can be expressed as

$$X^{KL\prime} = X_f + X_p(1) + \sum_{m=2}^{M}(X_t(m) + X_p(m)) + \sum_{i=1}^{L^H} X_h(i)$$

where $L^H$ is the number of HACKs received at the sender. Taking expectations we get

$$E[X^{KL\prime}] = E[X_f] + E[M]E[X_p] + (E[M] - 1)E[X_t] + E[L^H]E[X_H].$$

The expected number of HACKs received by the sender can be expressed as:

$$E[L^H] = E[M]B(1 - p)$$

where $B$ is the number of children of the sender and $E[M]$ is the expected number of transmissions needed to deliver a packet to a receiver. Substituting into $E[X^{KL\prime}]$ we get

$$E[X^{KL\prime}] = E[X_f] + E[M]E[X_p] + (E[M] - 1)E[X_t] + E[M]B(1 - p)E[X_H].$$

It can be shown (see [28] for details) that

$$E[M] \in O(1 + pln(R)/(1 - p)).$$

Substituting $B$ for $R$ in $E[M]$ we get

$$E[M] \in O(1 + pln(B)/(1 - p)).$$

This implies that

$$E[X^{KL\prime}] \in O(B(1-p) + Bpln(B)).$$

Notice that if $p$ is constant, then $E[X^{KL\prime}]$ is $O(Bln(B))$. In addition, if $p \longrightarrow 0$, then $E[X^{KL\prime}] = O(1)$.

**Receiver**

The expected processing time at the receiver will not change under $KL\prime$ since the group size does not have to change, and neither do the recovery schemes. In addition, as far as the receiver is concerned, the local repair server is indistinguishable from a sender. Therefore,

$$E[Y^{KL\prime}] \in O(1).$$

## 5.7 Grouping of Repair Servers

One possible combination of Local Recovery along with the concept of Grouping results in further improved processing requirements on the sender as will be evidenced below. In this scheme, the repair servers themselves form buddy groups for the purpose of error recovery just like receivers did under $K\prime$ and $KL\prime$. The local receivers of each repair servers also form buddy groups and recover from their own repair server. If that repair server does not have the missing packet, then it consults its repair server buddy group. If that group in turn does not have the missing packet, then $S$ is consulted. We call this new scheme the $KLL\prime$ protocol.

### 5.7.1 The $KLL\prime$ Protocol

The $KLL\prime$ protocol exhibits the following behavior:

- Receivers under this protocol act in the same way as under $K\prime$ except that the sender is now replaced with the local repair server. The rest of the steps below detail what happens during the grouping of repair servers.
- Upon joining a multicast group (session), a repair server $RS_i$ initially pairs up with the sender $S$ until a *group* $G$ of $n$ repair servers is formed.
- $RS_i$ informs $S$ that it is looking for a group.
- $S$ saves the address of $RS_i$ in a queue and waits until enough repair servers request grouping.
- Upon receiving a request for grouping, $S$ checks if it has $(n-1)$ repair servers waiting for grouping. If so, then $S$ informs $RS_i$ and and all other $(n-1)$ repair servers in the queue that all of them are going to be buddies in the same group for the purpose of error recovery. At that moment $RS_i$ drops its pairing with $S$ and groups with its new buddies in group $G$. If there are no repair servers waiting for grouping, $S$ behaves as in the previous step.
- All repair servers in $G$ now use point-to-point communication for error correction in a NACK-based fashion. When group members detect lost packets, they use one of the Group Recovery Schemes described earlier to recover the lost packets.
- Upon leaving a multicast session, a repair server informs its group members that it intends to leave. The group members then break the group and act as if is they have just joined the session, i.e. they pair with $S$ until enough members are found to form a new group.
- If some repair server in a group is not responding, then its buddies in the group can detect this either by polling it periodically, or by expiring timeouts, in which case they behave as in the previous step.

- If a repair server $RS_i$ has been dropped (dumped) by its group members, then upon reestablishing a contact with any of the former members, $RS_i$ is informed that it needs to find another group. $RS_i$ then behaves as if it's joining anew.

### 5.7.2 Performance Analysis of $KLL\prime$

**Sender**

If the repair servers themselves are grouped using the $K\prime$ protocol, then the end-to-end loss probability becomes $p^n$ both for receivers and repair servers since they are being grouped together $n$ receivers/repair servers at a time. Furthermore, the number of children of the sender can now be regarded as $B/n$ and receivers as $R/n$. Substituting in the expression for $E[M]$ given in the analysis section of $KL\prime$ we get a new expression for expected performance of the sender under $KLL\prime$ given by

$$E[X^{KLL\prime}] \in O(\frac{B}{n}(1-p^n) + \frac{B}{n}p^n ln(\frac{B}{n})).$$

**Receiver**

The expected processing time at the receiver will not change under $KLL\prime$ since the group size does not have to change, and neither do the recovery schemes. In addition, as far as the receiver is concerned, the local repair server is indistinguishable from a sender. Therefore,

$$E[Y^{KLL\prime}] \in O(1).$$

## 5.8 Summary

In this chapter we extended the concept of pairing from two to $n$ receivers. Those receivers formed groups for the purpose of error recovery. They only consulted the server when none of them had the lost packet. This reduced the probability of packet

loss to $p^n$ per group, and the session now consisted of $R/n$ groups. The concept of grouping was utilized in a new protocol that was described in detail. We called the new concept the $KI$ protocol. We also described three strategies for members of a group to recover their lost packet. Finally, we performed some analysis on the sender's and receiver's processing requirements under the $KI$ protocol. Just like the case of Global Pairing ($n = 2$), we maintained a constant complexity on the receiver's processing requirements. As for the sender, the complexity can easily compete with an $O(ln(R))$ protocol for small $p$ and for very small group sizes, generally under 10. The larger the group size, the smaller $p^n$ gets. As a result, the processing requirement at the sender is reduced. This provides for high tunability that allows you to shift the burden of processing away from the sender and into the receivers, without the receivers being dependent on the size of the multicast session. Finally, we combined the concepts of Local Recovery and Grouping and utilized it in a new protocol called the $KLI$ protocol. We then extended the concept of grouping to the repair servers themselves, while maintaining grouping at the receiver level. We utilized that concept in the $KLLI$ protocol. This achieved even further improvement on the sender's processing requirement, while maintaining a constant requirement on the receivers.

# Chapter 6

# Preemptive Multicasting

What happens when failures occur early on in the multicast tree, especially at or around $S$? Obviously, most if not all of the receivers will not have gotten the packet! Only if there was a way to determine that this happened, then $S$ would re-multicast the same packet in anticipation of a flood of NACKs or repair requests, whether from repair servers or from receivers themselves.

In this chapter, we introduce a new concept called *Preemptive Multicasting* whose objective is to predict if a large number of receivers, possibly all, never received a certain packet. It is done using a method similar to *statistical sampling.* If a certain threshold is met, then the packet is multicast again to all receivers.

## 6.1 Preemptive Multicasting

In order to determine if a packet was not received by a fairly large number of receivers, we need to be able to poll a representative sample of the receiver space. The problem is how to determine which receivers are part of the multicast session and which of them the sender is to poll. The set of $N$ receivers that $S$ decides to poll must fairly represent failure in the whole population space. Here is where the $K$ protocol comes into play.

### 6.1.1 Choosing a Random Sample

In the $K$ protocol, receivers requesting pairing are assumed to follow a random distribution with respect to the time they request pairing from the sender or with respect to their location in the network. The sender can randomly remember $N$ of

those receivers as the random sample and poll them. The sender, after originally multicasting each packet, requests a positive acknowledgement (ACK) from all N receivers in the polling sample. If all $N$ receivers reply negatively (that they did not receive the intended packet), or if none reply, then this is alarming enough to re-multicast the packet. Another case where re-multicasting should be considered is when the percentage of the number of negatively responding receivers exceeds a certain threshold $\tau$. The choice of $\tau$ is discussed in the next section.

### 6.1.2 Choosing a Threshold

Let the number of positively responding polled receivers be $P$, where $0 < P < N$. Let $\tau$, where $0 < \tau < 1$, be a multiplicative coefficient of $N$ such that a necessary condition for re-multicasting is

$$P \leq \tau N.$$

The key in determining the most accurate value of $\tau$ lies in the fact that even if all but one of the $n$ buddies forming a repair group fails to receive a packet, the group will self-heal and all $(n-1)$ will eventually recover the missing packet from the one that had it. We conclude then that $\tau = 1/n$ is the percentage of receivers that must receive the packet before re-multicasting is necessary. This is true since each group of $n$ buddies itself actually represents a random sample of the whole space (due to the random nature of grouping at the sender). The necessary condition for re-multicasting is now

$$P \leq N/n$$

where $n \geq 2$ and $N \geq n$. For the above condition to have any meaning, *the size of the polled sample of receivers $N$ must be at least as large as the number of buddies in a group.*

### 6.1.3 Choosing a Sample Size

The last concern is how big $N$ should be. Obviously, the larger $N$ is the more accurate the representation of the population of receivers. The two major issues to deal with here are the processing capability of the sender and the effect of polling on congestion. We believe that $N$ is a tunable parameter and it should be left up to the sender to determine the size of $N$, depending on any previous knowledge of its processing capability or of any congestion on the network. This really provides a certain degree of flexibility in tuning the performance of the protocol to control congestion at the server and in the network. The relationship between the size of $N$ and congestion should be studied in more detail and will be left for future research.

### 6.1.4 Maintaining the Polling Sample

One final challenge is to make sure that all or most of the $N$ receivers to be polled are still in the session and are still alive. As we have determined in a previous section, $N$ should be at least as large as $n$, the size of a buddy group. One possible solution is for $S$ to periodically poll the $N$ receivers and make sure that they are still alive. If it is determined that some receivers are not responding, then $S$ can drop them from the list and replace them with newer receivers that will request pairing in the future. This is one reason why $N$ should be considerably larger than $n$, in order to allow a buffer of receivers to be dropped without hurting the representation of the sample. Should $N$ drop below $n$, then the protocol may elect to turn off the preemptive multicasting feature until $N$ grows back to a healthy size, or solicit replacements for missing receivers. The latter can be done by sending a *solicitation-for-informers* message to all receivers. $S$ can then randomly choose enough receivers from those that respond to the solicitation message to fill the deficiency in the size of $N$.

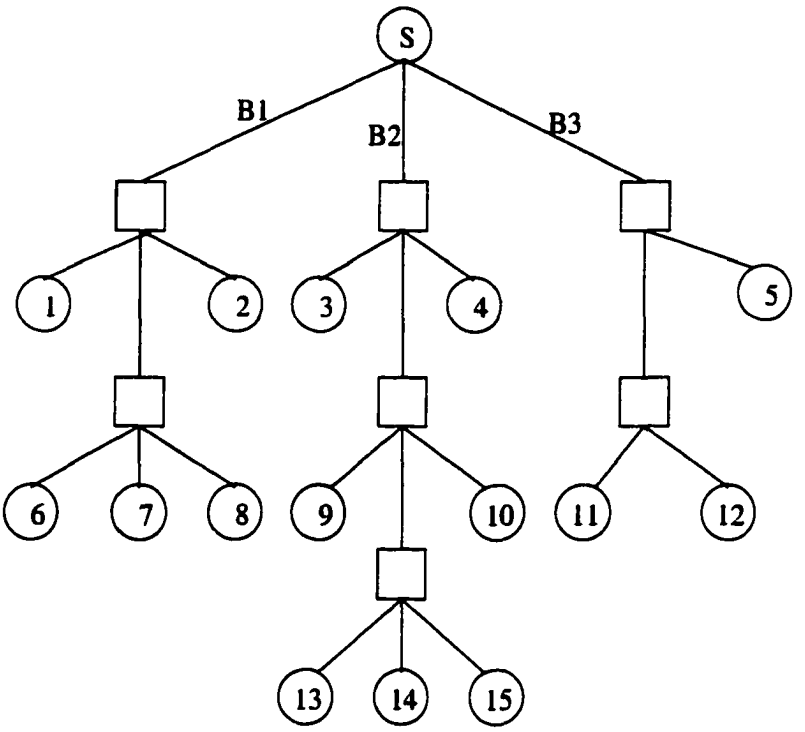
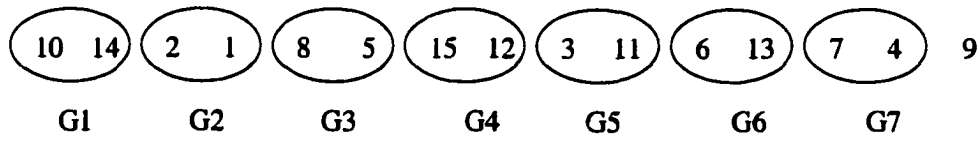



Figure 6.1: Preemptive Multicasting Example Tree

## 6.2 Illustrative Example

In this section we show an example of how preemptive multicasting helps reduce congestion at the sender resulting from an implosion of repair requests. Let us consider the topology in Figure 6.1 which represents the multicast delivery tree of some multicast session. The figure only depicts nodes that are members of the multicast session. This snapshot session consists of one sender $S$ and fifteen receivers, numbered 1 through 15. The network has three branches $B1$, $B2$ and $B3$ that feed data from $S$ to all other branches of the tree. The squares represent routers in the network.

Under the $K$ protocol, pairing requests arrive randomly at $S$. Buddy groups of size $n$ are then formed. For this small size example session, let us have $n = 2$. First, we generate a random distribution of receivers requesting pairing from $S$. Then we




PS = {1, 4, 6, 12} N = 4 n = 2

Figure 6.2: Random Distribution, Grouping and Polling Sample

group receivers two at a time to form seven buddy groups $G1$ through $G7$, plus one receiver (receiver 9) that is left unpaired. Under $K$, receiver 9 will actually pair up with $S$. Let us consider a polling sample (PS) size of $N = 4$. The random distribution of receivers and the corresponding grouping along with the random polling sample chosen by $S$ are all depicted in Figure 6.2.

To see the benefits of preemptive multicasting, let us consider the case where packet loss takes place at both $B1$ and $B3$, where none of the receivers under those branches receives the original multicast packet. That is eight out of fifteen, or 53% of the total number of receivers. Upon polling receivers in $PS$, only receiver 4 responds favorably. This meets the threshold to remulticast that same packet, which costs $S$ on extra sending of the packet. Had remulticasting not been done, groups $G2$ and $G3$ would not have been able to recover the lost packet since receivers 1 and 2 of $G2$ fall in the same branch $B1$ where the packet failed to be delivered. Similarly, receivers 8 and 5 of $G3$ fall in two different branches that failed. As a result, they would not be able to recover the lost packet. In both cases under $K$, one member from each group will end up requesting repair from $S$. In this case $S$ will send two repair packets, one to each of the two groups. This costs $S$ one more multicast of the lost packet.

Now consider the case where the packet is lost at $B2$ and $B3$. That is 75% of the total number of receivers failing to receive the packet. Upon polling receivers in

$PS$, only 1 and 6 respond favorably (assuming that both of them actually received the packet), which causes remulticasting to take place. Had remulticasting not been done, groups $G1$, $G4$, $G5$ and receiver 9 would not have been able to recover the lost packet. In that case, each of the three groups and receiver 9 would have requested repair from $S$ resulting in three more repair packets to be sent! Now we can easily see the savings that preemptive multicasting would have resulted in.

One last case to consider shows that preemptive multicasting is not always perfect due to the fact that it depends on statistics in determining whether to remulticast or not. Consider the case when packet loss takes place along $B2$. This causes $G1$ and receiver 9 to request repair from $S$ resulting in one more repair packet to be sent by $S$.

## 6.3 Effect of Sample Polling

Here we try to analyze the effect of sample polling on the sender $S$ and on the network and receivers.

### 6.3.1 Effect on Sender

Per multicast packet, $S$ has to perform the following:

- Send $N$ polling messages to $N$ receivers
- Process at most $N$ ACKs
- Possibly process one timer before giving up on polling
- Preparation and re-multicast of the packet if necessary

All of the above tasks combined will not be significant unless $N$ grows to be very large. For small values of $N$ the sender will not suffer as a result of sample polling.

### 6.3.2 Effect on Receivers

Per multicast packet, only $N$ out of $R$ receivers will perform the following:

- Process polling request from $S$

- Respond to $S$

Since the polled sample of receivers is built by caching the last $N$ receivers that request pairing, the sample can be thought of as a *sliding window* of receivers, where any receiver $R_i$ stays in the polling sample only from the time it requests pairing until $N$ more receivers request pairing, in which case it is dropped out of the window. While being in the window, executing the two tasks mentioned above requires a constant and insignificant amount of processing. Therefore, it will not have any severe effect on the processing capability of the polled receiver. Naturally, all other $(R - N)$ receivers are not affected by the polling procedure.

### 6.3.3 Effect on the Network

Per multicast packet, the network may be injected with the following added traffic as a result of polling:

- $N$ polling messages to $N$ receivers
- $N$ ACKs from receivers
- one re-multicast of the packet if necessary

Each network link in the multicast subtree leading to the $N$ polled receivers will have the added traffic of only two extra control packets (these packets are typically very small), and possibly one re-multicast of the missing packet. Again, if $N$ is sufficiently small constant, then the effect on network bandwidth is negligible.

## 6.4 Summary

Preemptive multicasting should be thought of as an aiding mechanism in loss recovery. It should not be thought of as the sole and primary error correction mechanism, but rather as a helper feature that cooperates with a well defined error detection and recovery protocol in order to avoid implosions from repair requests. Preemptive multicasting is highly useful especially when the probability of failure is high or if

losses occur in the higher levels of the multicast delivery tree. In that case many receivers will not have received a packet and preemptively remulticasting it by $S$ would reduce congestion on the network. As we have seen above, preemptive multicasting does not have any significant effect on the processing requirements of either the sender or the receivers. In addition, it only injects the network with a negligible amount of extra traffic. Finally, preemptive multicasting can be implemented as a tunable feature on demand in any reliable multicast transport protocol for the purpose of reducing congestion on the network.

# Chapter 7

# Performance Comparison

In this chapter we summarize the results we obtained from the performance analysis on the $K$ family of protocols. We then compare the improvement in processing requirements separately on the receiver and on the sender side. Since the results on the receiver were very consistent, we will start our discussion with the receiver. However, Let us first take a look at a summary table of our results, presented in Table 7.1.

## 7.1 Receiver

The $K$ family of protocols consistently maintained a *constant* processing requirement complexity throughout, which is a formidable task since it affects $R$ receivers, where $R$ could be thousands or hundreds of thousands or even millions of hosts on the Internet. Regardless of how low the complexity of the sender is, a receiver should not suffer because a multicast session grew very large. That is the complexity of the

Table 7.1: Summary of $K$ Protocol Family Results

| Protocol | Sender Requirements | Receiver Requirements |
|---|---|---|
| $K$ | $O(1+\frac{p^2}{2}R)$ | $O(1)$ |
| $KL$ | $O(B(1-p)+Bpln(B))$ | $O(1)$ |
| $K'$ | $O(1+\frac{p^n}{n}R)$ | $O(1)$ |
| $KL'$ | $O(B(1-p)+Bpln(B))$ | $O(1)$ |
| $KLL'$ | $O(\frac{B}{n}(1-p^n)+\frac{B}{n}p^n ln(B/n))$ | $O(1)$ |

receiver should not be of any order of $R$, not even logarithmic or sub-logarithmic. In our $K$ protocols, we have shifted some of the burden of processing either to the sender, to the repair servers, or to the backbone (communication between buddies is an example of this.)

One major advantage of using one of the protocols of the $K$ family is that the receiver need not setup and maintain a multicast tree to all other receivers in the session. The task of multicasting NACKs to all the receivers, as is the case in the Receiver-initiated and RINA family of protocols, is not trivial since it involves the setup and maintenance of a multicast delivery tree. The dynamic nature of a multicast session, with receivers joining and leaving frequently, makes this task an unfavored one and it should be avoided when necessary. This corroborates the need for a family of protocols that does not involve receivers in tasks that are of the order of the whole session size. The $K$ family of protocols clearly achieves this objective effectively.

## 7.2 Sender

There is more to the complexity of the sender under the $K$ protocol than meets the eye. This is due to the $p^n/n$ factor of $R$. Since $0 \leq p \leq 1$, $p^n$ will shrink exponentially for smaller values of $p$ and larger values of $n$. The size of the buddy group plays a very important role in how much processing the sender has to do. This is because the larger $n$ is, the less processing $S$ has to do. The value of $n$ can be used as a tunable parameter in any implementation of the $K$ family since it allows you to determine, based on estimated multicast session size, how large $n$ should be to meet a certain processing demand on the server. This is a very powerful concept in tuning a multicast session to the requirements of the server. If fact, $n$ could be a parameter that $S$ determines and forces it upon receivers.

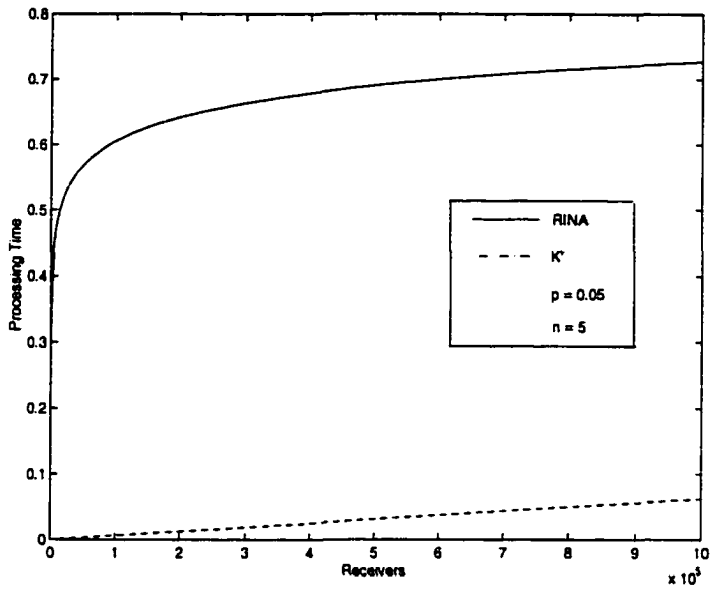



Figure 7.1: RINA versus $K\prime$, p=0.05, n=5

In Figures 7.1 through 7.9 we have permuted the values of $n$ and $p$ for low, medium and high values, producing nine graphs. The graphs are plots of the complexity of the RINA class of protocols versus $K\prime$. The value of $R$, the total number of receivers in the session, has been set in the range from 1 to 1 million.

We observe that in the cases where $p$ was high or $n$ was low, or both, the RINA class of protocols outperformed $K\prime$ (Figures 7.2, 7.3 and 7.9). However, when $p$ was low and $n$ relatively higher, the $K\prime$ protocol outperformed RINA.

Therefore, we conclude that the $K\prime$ protocol can match or even outperform the RINA class of protocols by setting $n$, the size of the buddy group, to an appropriate value, for an estimated session size. As we have seen from the graphs, we could do this for one million receivers, with as little as 5 and as high as 10 members in the buddy group.

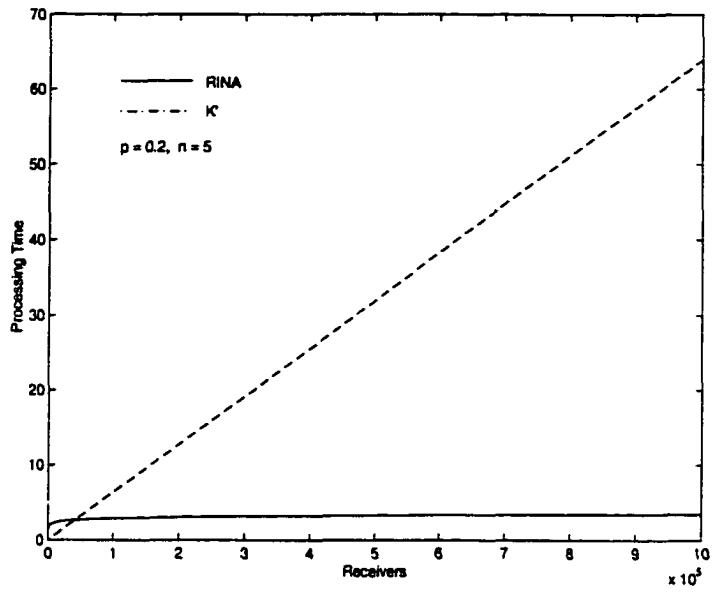



Figure 7.2: RINA versus $K'$, p=0.20, n=5

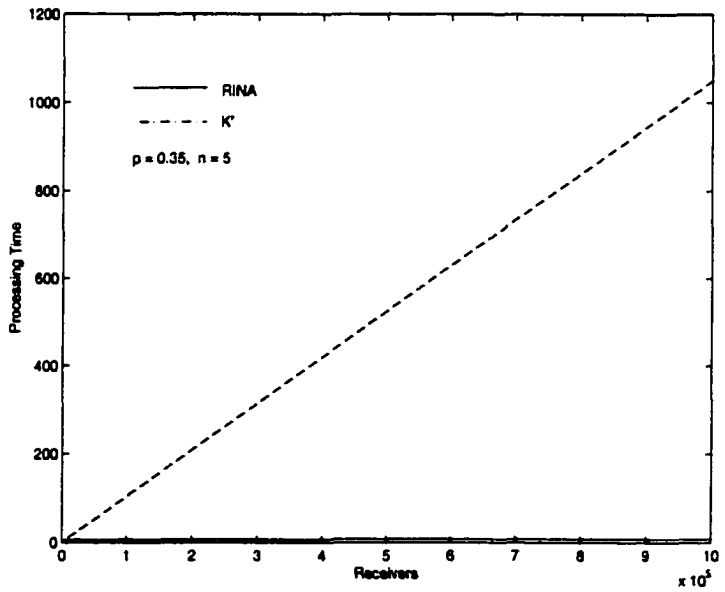



Figure 7.3: RINA versus $K'$, p=0.35, n=5

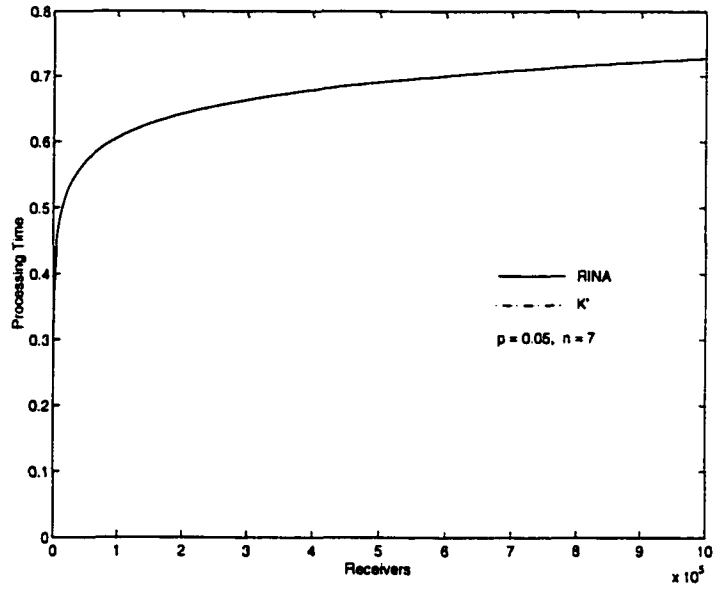



Figure 7.4: RINA versus $K'$, p=0.05, n=7

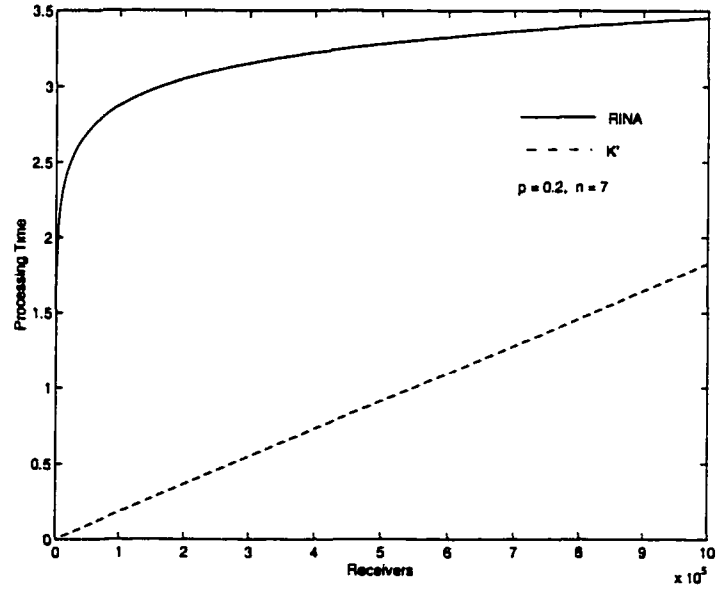



Figure 7.5: RINA versus $K'$, p=0.20, n=7

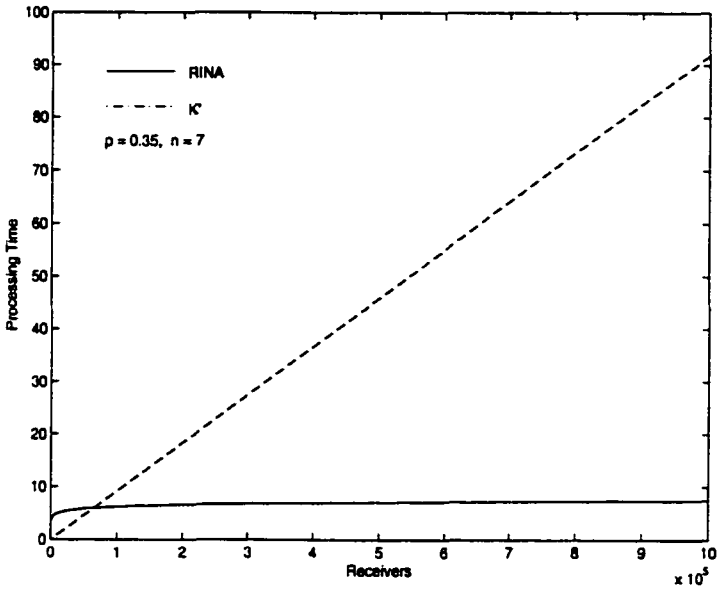



Figure 7.6: RINA versus $K'$, p=0.35, n=7

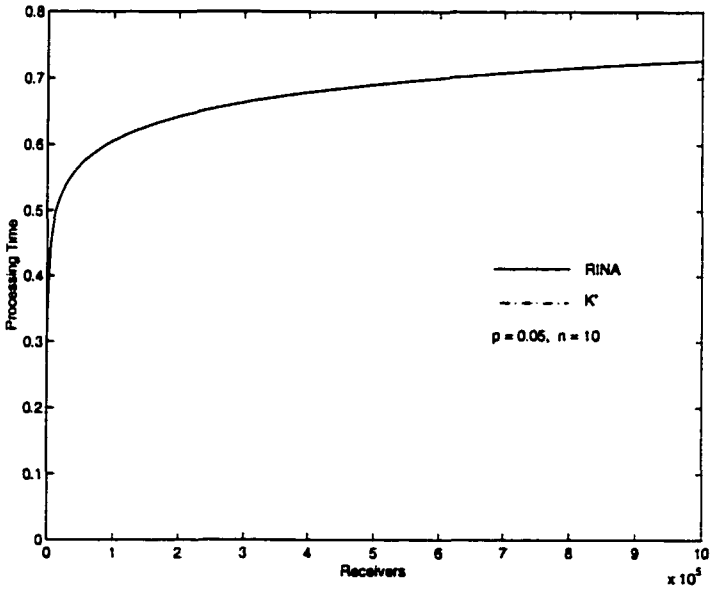



Figure 7.7: RINA versus $K'$, p=0.05, n=10

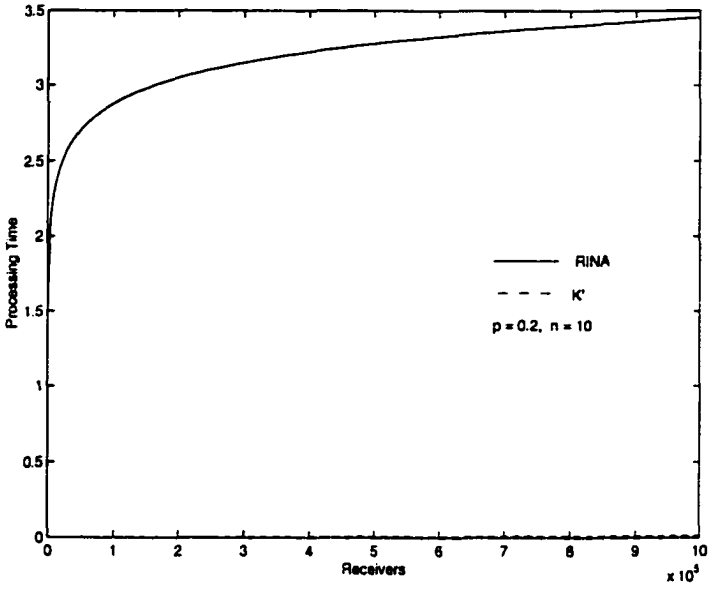



Figure 7.8: RINA versus $K'$, p=0.2, n=10

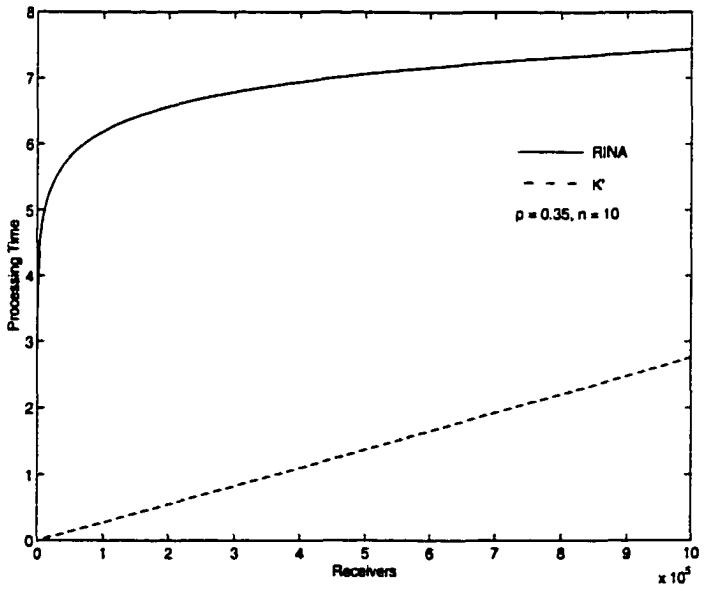



Figure 7.9: RINA versus $K'$, p=0.35, n=10

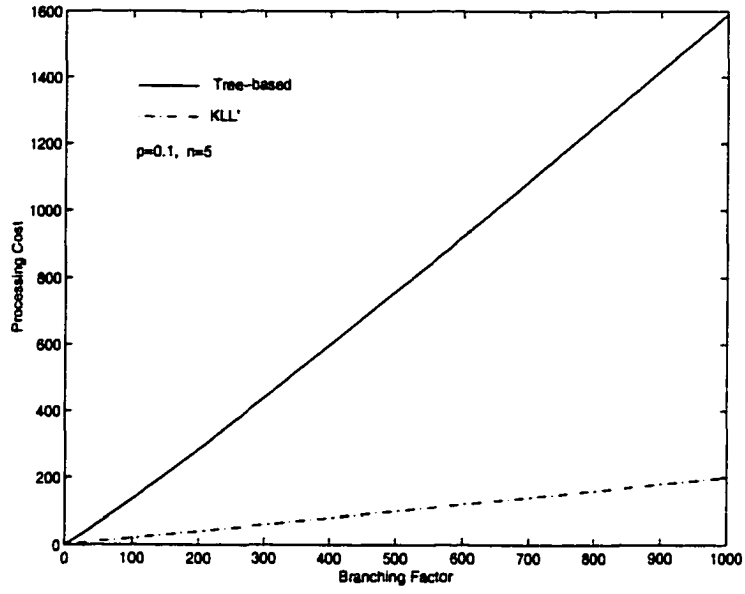



Figure 7.10: Tree-Based versus *KLL'*, p=0.1, n=5

Finally, in Figure 7.10 we see another plot of the complexity of protocol *KLL'* versus that of Tree-based protocols. You can immediately see the reduction in processing cost, even for small values of $n$ and $p$.

## 7.3 Comparison with Other Methods

In order to see how our results improve on the results of other methods, we must compare them with similar-class methods. For example, we should compare our protocols that use local recovery with other methods that use local recovery. Let us begin by looking at Table 7.2 where we compare the complexity of our *K'* protocol (that does not utilize any form of local recovery) with those of Sender-Initiated, Receiver-Initiated and RINA classes of protocols.

We can clearly see that *K'* outperforms the Sender-Initiated and Receiver-Initiated classes of protocols. Also, under the conditions described earlier, *K'* has the potential of outperforming the RINA class of protocols, especially by tweak-

Table 7.2: Comparison of $KI$ with Sender and Receiver-Initiated Methods

| Protocol | Sender Requirements | Receiver Requirements |
|---|---|---|
| Sender-Init. | $O(R(1+pln(R)/(1-p)))$ | $O(1-p+pln(R))$ |
| Receiver-Init. | $O(1+pR/(1-p))$ | $O(1-p+pln(R))$ |
| RINA | $O(1+pln(R)/(1-p))$ | $O(1-p+pln(R))$ |
| $KI$ | $O(1+\frac{p^n}{n}R)$ | $O(1)$ |

Table 7.3: Comparison of $KLI$ and $KLLI$ with Local Recovery Methods

| Protocol | Sender Requirements | Receiver Requirements |
|---|---|---|
| Tree-Based | $O(B(1-p)+pBln(B))$<br>DR: $O(B(1-p)+Bpln(B))$ | $O(1-p+pln(B))$ |
| $KLI$ | $O(B(1-p)+pBln(B))$ | $O(1)$ |
| Tree-NAPP | $O(1+pln(B)/(1-p))$ | $O(1+((1-p+pln(B)+$<br>$+p^2(1-4p))/(1-p))$ |
| $KLLI$ | $O(\frac{B}{n}(1-p^n)+\frac{B}{n}p^n ln(B/n))$ | $O(1)$ |

ing the size of the buddy group $n$ to achieve the desired reduction in processing requirement.

Now let us compare our protocols that utilized local recovery and grouping with those that utilized local recovery in other classes of protocols, namely Tree-Based and Tree-NAPP. This comparison is depicted in Table 7.3. There is no similarity to our grouping mechanism in any other class of protocols, so it stands unique in that aspect. However, we have achieved equivalent complexity in the case of $KLI$ and Tree-Based protocols, with the added advantage of maintaining constant complexity on the receivers. As for the $KLLI$ protocol, it's clear that it outperforms Tree-NAPP protocols while also maintaining constant complexity on the receivers.

## 7.4 Observations

Why did we compare the $K\prime$ protocol only to RINA when some other protocols exhibited lower complexities (even constant ones) on the part of the sender and/or the receiver? The answer lies in the class of protocols we are dealing with as well as any *extras* that have been added at some cost. For example, in the class of Ring-Based protocols the sender and receiver are both of the order $O(1)$. That does not mean that the protocol is more efficient than ours or RINA because it comes at a cost. Arranging the receivers (all $R$ of them!!) in the form of a ring seems impossible due to the dynamic nature of a multicast session. Every time a receiver joins or leaves the session the ring must be updated. In addition, one of the receivers at any instance of time is designated a *token site*, and it becomes responsible for sending ACKs back to the sender. The token site also retransmits missing packets. This seems too much to do for an average receiver in the session. In fact the token site's complexity is computed at $O(1 + p(R-1)/(1-p))$.

Another case is that of Tree-Based protocols that exhibit an $O(B(1-p) + Bpln(B))$ complexity. $B$ is the branching factor of the sender, or its number of children. This seems good at the surface since it's independent of $R$, but there is the cost of the designated routers, or repair servers. This is not trivial in an Internet that consists of millions of hosts. Any neat protocol should take into consideration feasibility without incurring heavy costs, materially or computationally.

# Chapter 8

# Conclusions

This research effort focused on congestion control mechanisms for Internet Multicast Transport protocols. More specifically, we focussed on the following areas:

1. Reducing the processing requirements on the receiver to a constant.
2. Reducing the processing requirements on the server for the same class of protocols, namely receiver-initiated with no special arrangement of receivers or repair servers.
3. Devising a new class of protocols to achieve the above targets by introducing the new concepts of Pairing and Grouping utilized in the $K$ family of protocols that we described.
4. Applying the concept of Local Recovery to the $K$ family of protocols to achieve even greater performance on the server.
5. Devising a new concept called Preemptive Multicasting that attempts to predict heavy packet loss in the network and remedy it before any repair request flood the network.

Chapters 3, 4, 5 and 6, respectively, discussed the issues mentioned above in detail and provided new protocols and algorithms.

## 8.1 The One-Size-Fits-All Protocol

It has been generally agreed upon that there is no one-size-fits-all multicast protocol that is suitable for all applications. Several proposals have been made to shift the burden of framing data units even to the application level. This approach has still a

long way to go, especially in finding a way to address data units at that level. Until then, there is still room for improvement at the transport level. Congestion control is one of the major issues to deal with when designing efficient multicast transport protocols. This is what we have attempted to do in this research effort. Our new family of protocols introduced a very novel and new concept, that of grouping. Other protocols in the past have limited receivers to recovering their losses either directly from the server or from some *unknown* receiver. Now receivers have someone else to check with – another well-known receiver. This knowledge was not available in the past without broadcasting and polling, which took a toll on the network. The concept of grouping took advantage of receivers actively seeking partners or buddies for the purpose of error recovery. The random nature of their requests allowed the server to group them exactly in that fashion, which proved to be very efficient in many aspects. Applying local recovery concepts further improved the performance of end hosts.

## 8.2 Future Directions

There is a lot left to do in the area of Multicasting, especially in multicast transport protocols. A natural extension of our research would aim at simulating our protocols in different topologies and under different probabilities of loss and group sizes. In addition, studying the effect of grouping on bandwidth would be of paramount interest if we relax our assumption about bandwidth not being a point of contention for some time to come. If this changes, then how do our protocols adapt?

Finally, we must build multicast enabled applications that take advantage of our research results and other results as well. The MBone is a great place to start, but it will certainly take much more than that to migrate the Internet from its current state to using multicast as an efficient way to deliver replicated data in the

future. All vendors must agree to some multicast standard and implement it in their network hardware. Some protocols and standards must be adopted soon for this to take off. It is up to each one of us to decide whether to pitch in or not. We hope to have done so in this research effort.

# Bibliography


[1] P. Bhagwat, P. Misra and S. Tripathi, "Effect of Topology on Performance of Reliable Multicast Communication," *Proc. IEEE Infocomm 94,*Toronto, June 1994, pp. 602-609.

[2] R. Braudes and S. Zabele, "Requirements for Multicast Protocols," RFC 1458, May 1993.

[3] R. Buskens, M. Siddiqui and S. Paul, "Reliable multicasting of continuous data streams," *Bell Labs Technical Journal,*vol. 2, no. 2, 1997, pp. 151–174.

[4] Jo-Mei Chang and N. F. Maxemchuk, "Reliable Broadcast Protocols," *ACM Transactions on Computer Systems,* Vol. 2, No. 3, August 1984, pp. 251–173.

[5] D. Clark and D. Tennenhouse, "Architectural considerations for a new generation of protocols," *Proc. ACM SIGCOMM 90,* Sept. 1990, pp. 353–359.

[6] S. Deering, "Multicast Routing in a Datagram Internetwork," Ph.D. Dissertation, Stanford University, Palo Alto, CA, December 1991.

[7] A. Erramilli and R. P. Singh, "A Reliable and Efficient Multicast Protocol for Broadband Broadcast Networks," *Proc. ACM Sigcomm 88,* pp. 343-352, August 1988.

[8] S. Floyd, V. Jacobson, C. Liu, S. McCanne and L. Zhang, "A Reliable Multicast Framework for Light-Weight Sessions and Application Level Framing," *IEEE/ACM Transactions on Networking,* Vol. 5, No. 6, December 1997.

[9] M. Fuchs, C. Diot, T. Turletti and M. Hofmann, "A Naming Approach for ALF Design," *Third International Workshop on High Performance Protocol Architectures* (HIPPARCH), London, England, June 15-16, 1998.

[10] M. Goncalves and K. Niles, *IP Multicasting Concepts and Applications,* McGraw-Hill, 1999.

[11] B. Graham, "TCP/IP addressing; designing and optimizing your IP addressing scheme," Academic Press, San Diego, CA, 1997.

[12] M. Hofmann, "A generic concept for large-scale multicast," *Proceedings of the International Zurich Seminar on Digital Communications,* (IZS 96), February 1996.

[13] H. W. Holbrook, S. K. Singhal and D. R. Cheriton, "Log-Based Receiver Reliable Multcast for Distributed Interactive Simulation," *Proceedings of ACM SIGCOMM*, pages 328–341, August 1995.

[14] C. Huitema, "IPv6 The New Internet Protocol," Prentice Hall PTR, Upper Saddle River, NJ, 1996

[15] C. Huitema, "Routing in the Internet," Prentice Hall PTR, Englewood Cliffs, NJ, 1995.

[16] S. Kasera, J. Kurose and D. Towsley, "A Comparison of Server-Based and Receiver-Based Local Recovery Approaches for Scalable Reliable Multicast," CMPSCI Technical Report TR 1997-69, Department of Computer Science, University of Massachusetts, Amherst, MA, December 1997.

[17] V. Kumar, "MBone: Interactive Multimedia on the Internet," New Riders Publishing, Indianapolis, IN, 1996.

[18] B. N. Levine and J. Garcia-Luna-Aceves, "A Comparison of Known Classes of Reliable Multicast Protocols," *Proceedings of ACM Multimedia*, November 1996.

[19] J. C. Lin and S. Paul, "RMTP: A Reliable Multicast Transport Protocol," *Proceedings of IEEE Infocom*, 1995.

[20] T. A. Maufer, *Deploying IP Multicast in the Enterprise,* Prentice Hall PTR, 1998.

[21] C. K. Miller, *Multicast Networking and Applications,* Addison Wesley, 1999.

[22] C. K. Miller, K. Robertson, A. Tweedly and M. White, "Multicast File Transfer Protocol (MFTP) Specification," Internet Draft, Work in Progress, draft-miller-mftp-spec-03.txt, April 1998.

[23] S. Paul, *Multicasting on the Internet and its Applications,* Kluwer Academic Publishers, 1998.

[24] S. Paul, K. K. Sabnani and D. M. Kristol, "Multicast Transport Protocols for High Speed Networks," *Proceedings of International Conference on Network Protocols*, pp. 4–14, 1994.

[25] S. Pingali, J. Kurose and D. Towsley, "A Comparison of Sender-Initiated and Receiver-Initiated Reliable Multicast Protocols," *Proceedings of ACM Sigmetrics Conference*, May 1994.

[26] W. T. Strayer, B. J. Dempsey and A. C. Weaver, "XTP: The Xpress Transfer Protocol," Addison-Wesley Publishing Company, Inc., 1992.

[27] A. Tanenbaum, "Computer Networks," Third Edition, Prentice Hall PTR, Upper Saddle River, NJ, 1996.

[28] D. Towsley, J. Kurose and S. Pingali, "A Comparison of Sender-Initiated and Receiver-Initiated Reliable Multicast Protocols, " *IEEE Journal on Selected Areas in Communications,* Vol. 15, No. 3, pp. 398–406, April 1997.

[29] B. Whetten, T. Montgomery and S. Kaplan, "A High Performance Totally Ordered Multicast Protocol," *Theory and Practice in Distributed Systems,*K. P. Birman, F. Mattern, A. Schiper (Eds), Springer Verlag LNCS 938.

[30] R. Yavatkar, J. Griffioen and M. Sudan, "A Reliable Dissemination Protocol for Interactive Collaborative Applications," *Proceedings of ACM Multimedia,* November 1995.

# Vita

Elias G. Khalaf received his bachelor of science degree in Computer Science from the University of Southwestern Louisiana (currently The University of Louisiana at Lafayette), Lafayette, Louisiana, in 1989. Since then, he assumed the position of Computer Manager at the department of Computer Science at Louisiana State University, where he is still employed. In 1990, he joined the graduate program at L.S.U. and received a master of science degree in System Science in 1992. Since then, he has been working part-time towards the degree of Doctor of Philosophy in Computer Science in the same department. He will receive his doctorate in 2000. He frequently taught Computer Science classes for the evening school at L.S.U. His research interests include Internet Multicasting, Distributed Computing and Networking.

**DOCTORAL EXAMINATION AND DISSERTATION REPORT**

**Candidate:** Elias G. Khalaf

**Major Field:** Computer Science

**Title of Dissertation:** Congestion Control Mechanisms for Internet Multicast Transport Protocols

**Approved:**

Major Professor and Chairman

Dean of the Graduate School

**EXAMINING COMMITTEE:**

**Date of Examination:**

December 6, 1999